Publicaciones Técnicas
pubtecia@astro.unam.mx

Instituto de Astronomía
UNAM

Reporte Técnico RT-2012-04

Aceptación: 14-08-2012
Recepción: 23-08-2011

Manejo vía TCP/IP de cámaras Finger Lake Instruments (FLI).

E. Colorado, D. Hiriart.
Instituto de Astronomía. Universidad Nacional Autónoma de México.
Km. 103 Carretera Tijuana-Ensenada, Ensenada, B. C., México.

RESUMEN:

Este documento describe la programación de control desarrollada para manejar las cámaras de la compañía "*Finger Lake Instruments*" (FLI) a través de un servidor TCP/IP. El programa fue desarrollado en el lenguaje de programación C/C++ y permite el control y la adquisición de imágenes vía internet. El programa es ejecutando en una computadora dedicada de la marca "*Sheevaplug*" de bajo costo y consumo eléctrico. Esto permite estandarizar el manejo de las cámaras FLI del OAN-SPM a través de la red de internet y facilitar su operación robótica y remota. El programa ha sido probado con éxito en la cámara FLI utilizada en los instrumentos POLIMA y MEXMAN del telescopio de 0.84 m del OAN-SPM.

Contenido

1. INTRODUCCIÓN

El Observatorio Astronómico Nacional en la sierra de San Pedro Mártir, B.C. (OAN-SPM) cuenta con varias cámaras de la compañía *"Finger Lakes Instruments"* (FLI) que se utilizan en los telescopios de 1.5 y 0.84 metros. Para estandarizar el manejo de las cámaras a través de la red de internet, desarrollamos un programa modular de bajo nivel en el lenguaje de programación C++ para el control y adquisición de imágenes. El programa se ejecuta en una computadora dedicada *"SheevaPlug"* de bajo costo y bajo consumo eléctrico. Esta computadora funciona como un puente entre la red de internet y la cámara FLI. La *"SheevaPlug"* recibe mandos TCP/IP a través de la red de internet, los convierte en mandos apropiados para las cámaras FLI y los envía a la cámara vía un puerto USB. Asimismo, la *"SheevaPlug"* hace accesible a la red las imágenes obtenidas y toda la información del estatus de la cámara. La interacción con la *"SheevaPlug"* se puede realizar desde cualquier computadora autorizada que esté conectada a internet. El desarrollo de este programa es de propósito general y se ha utilizado con éxito en el telescopio de 0.84 m del OAN-SPM en el polarímetro de imagen POLIMA que utiliza una cámara FLI de la serie ProLine PL3041-UV. En este documento se describe la programación desarrollada.

2. OBJETIVOS

El propósito de este paquete de programas es tener una plataforma común para el control de todas las cámaras FLI para permitir el control y la transferencia de imágenes vía red (protocolo TCP/IP) de manera que sea posible el uso remoto o robótico de la cámaras FLI.

3. DESARROLLO

En esta sección se describen los detalles técnicos del programa, así como la lista de mandos y parámetros que el programa recibe y manda a través de la red de internet utilizando el puerto 9710 del protocolo de comunicaciones de red TCP/IP.

El programa llamado "**fli_server2**" fue desarrollado para el sistema operativo Linux en C++ utilizando las librerías externas "**libfli-1.99**" de la compañía FLI para el manejo de bajo nivel de sus cámaras. Esta librería se puede descargar de la siguiente dirección electrónica: *http://www.flicamera.com/software/index.htm*.

3.1. EL PROGRAMA FLI_SERVER2

La *Figura 1* muestra el diagrama de flujo del programa "**fli_server2**". El programa realiza las siguientes funciones al iniciar:

- Carga nuestra clase en C++ FLI
- Abre un "*socket*" como servidor utilizando el puerto TCP/IP 9710.
- Entra en un ciclo de espera para recibir instrucciones, a través de la red de internet, de un cliente vía el protocolo TCP/IP.

- Al recibir una instrucción, el programa abre una hebra (cómputo paralelo) para ejecutar el mando recibido.

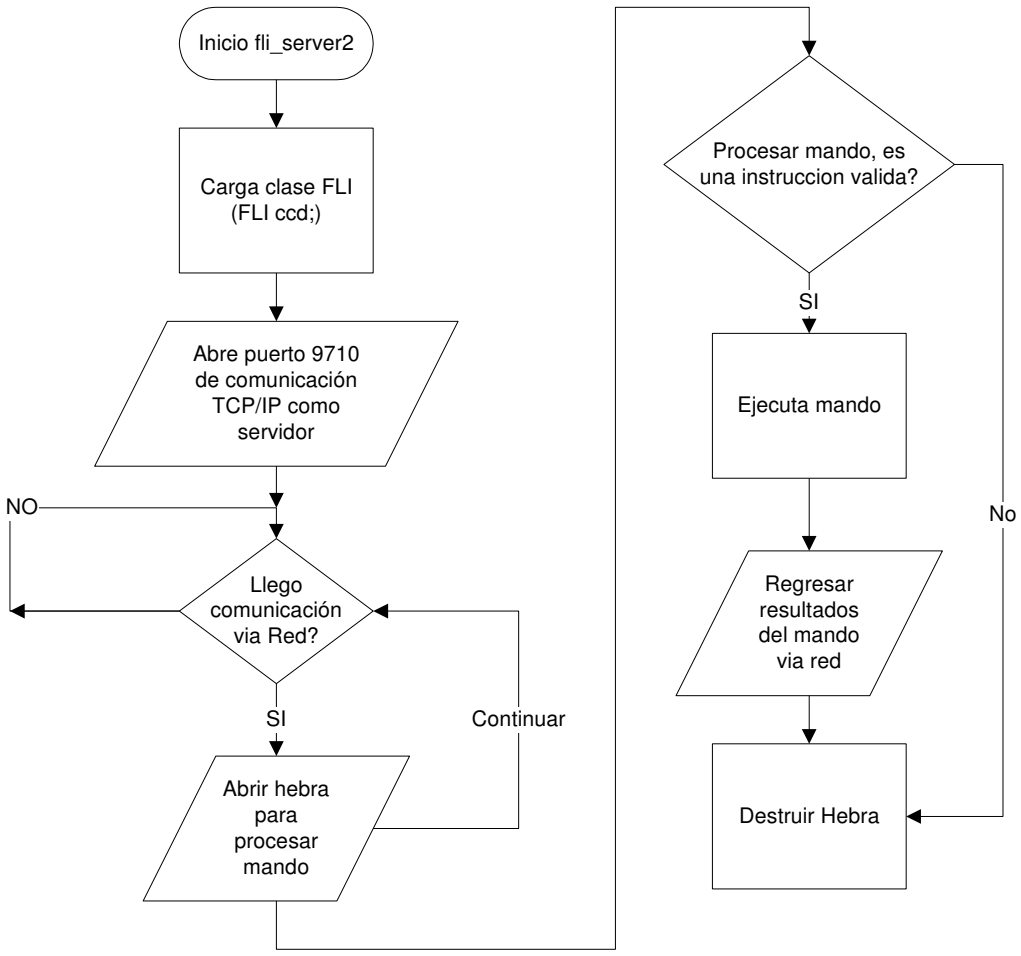


Figura 1: Diagrama de flujo principal.

La primera instrucción que el programa de control debe recibir es la instrucción de inicialización "INIT". La *Figura 2* muestra el diagrama de flujo del programa para la ejecución de esta instrucción. Las operaciones que realiza al recibir esta instrucción son las siguientes:

- Busca una cámara de la compañía FLI conectada el puerto USB y establece una conexión.
- Muestra el modelo de la cámara encontrada.
- Lee las dimensiones del dispositivo CCD instalado en la cámara.
- Reserva memoria para almacenar la imagen adquirida.
- Maneja la celda Peltier para controlar la temperatura del CCD.
- Regresa la información encontrada al cliente.

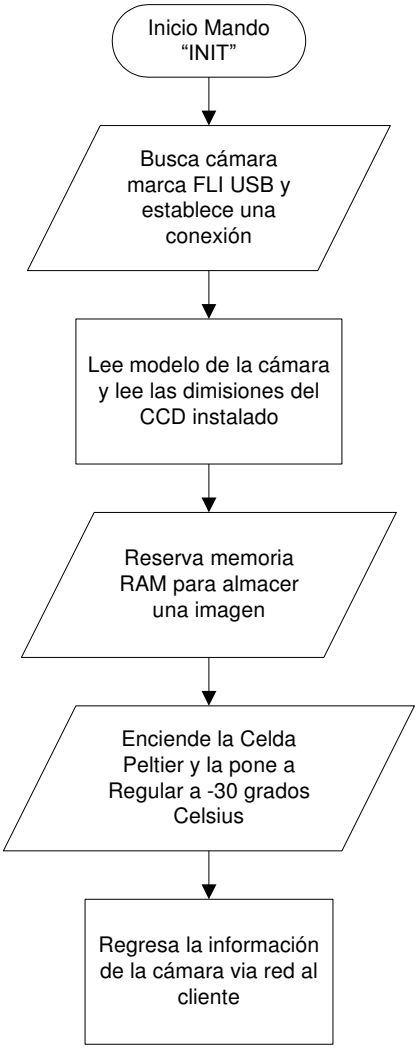


Figura 2: Diagrama de flujo de instrucción "INIT".

Por ejemplo, al recibir la instrucción vía red "*EXPONE ETIME=100 XSIZE=2048 YSIZE=2048 CBIN=1 RBIN=1 CORG=0 RORG=0 DARK=0*", se le está indicando al programa **fli_server2** que debe de tomar una imagen completa de 100 milisegundos y el programa realizará los siguientes pasos:

- Programará el tiempo de exposición a la cámara.
- Mandará a exponer una imagen al CCD.
- Capturará la imagen resultante y la almacenará en RAM.
- Leerá la temperatura del CCD, la temperatura externa y el porcentaje utilizado de la celda Peltier.
- Regresará al cliente conectado la leyenda "*BINARIO x*", Donde x = numero de pixeles leídos del CCD.
- Cierra el canal de comunicación con el cliente y espera una nueva conexión.

3.2. LISTA DE MANDOS VÍA RED

A continuación se listan los mandos aceptados vía red. Estos mandos siempre regresan al último la cadena de caracteres “CLOSE \n”

TABLA 1

Mandos vía red.

	Argumento	**Descripción**	**Regresa**
INIT		Busca las cámaras FLI conectadas a los puertos USB e inicializa la primera cámara encontrada.	info Library version: Software Development Library for Linux 1.99 info type: /dev/fliusbo, -->USB info Model: PL3041 info pixel size: 0.000015 x 0.000015 info Array area: (0, 0)(2080, 2112) info Visible area: (16, 32)(2064, 2080) info Work Size: 2048, 2048 LEE_TEMP -35.0 GOOD ->CCD INIT OK
CLOSE		Cierra la comunicación con la cámara.	
GET_TEMP		Lee la temperatura de la cámara.	LEE_TEMP x Donde x = temperatura de la cámara en grados Celsius.
SET_TEMP	X Donde x = temperatura deseada a regular de la cámara en grados Celsius.	Fija la temperatura a regular por la celda Peltier de la cámara.	

TABLA 2

Mandos vía red (Continuación)

LEE_TEMP_BASE		Lee la temperatura de la base de la cámara.	LEE_TEMP_BASE x donde x = temperatura de la base del CCD en grados Celsius.
LEE_POWER		Lee el porcentaje de potencia utilizada por la celda Peltier.	LEE_POWER x donde x = Porcentaje de utilización de la celda Peltier.
EXPONE	t xsize ysize binning xorg yorg shutter dark t =Tiempo en milisegundos. xsize= Tamaño de columnas de la imagen deseada. ysize=Tamaño de renglones de la imagen deseada. binning=Factor de binning, puede ser 1,2 ó 3. xorg =Columna inicial de la imagen. yorg =Renglón inicial de la imagen. shutter=0 ó 1. Con 1 se abre el shutter. dark=0 ó 1. Con 1 a la imagen resultante se le sustraerá automáticamente un dark.	Manda a exponer una imagen completa o parcial, pudiendo restar el dark.	TIME_REMAINING x BINARIO y LEE_TEMP z TERMINE Donde x = Milisegundos faltantes de exposición. y=Numero de pixeles adquiridos. z=Temperatura actual del CCD.

TABLA 3

Mandos vía red (Continuación)

MANDABIN		Manda vía el socket TCP/IP la última imagen guardada en RAM.	Este mando no regresa la palabra "CLOSE" al final.
STATUS		Pide el estado del servidor, si se está comunicando con la cámara.	Regresara BUSY o READY. Nota: No mandar otros comandos a la cámara si el estado es BUSY.
SET_CAM_MODE	X Donde x=0, si se desea la cámara a 2 MHz @ 2 canales. x=1, si se desea la cámara a 2 MHz @ 1 canal. x=2, si se desea la cámara a 500 Khz @ 1 canal.	Cambia el modo de lectura del CCD.	
GET_CAM_MODE		Regresa el modo de lectura actual. Ver mando anterior.	CAM_MODE x Donde x= Modo de lectura actual.
GET_CAM_STRING		Regresa una cadena de texto con la información de la cámara encontrada.	CAM_MODE_STRING x Donde x= Cadena de caracteres indicando el modo de lectura actual.
SALIR		Termina la ejecución del programa servidor.	

3.3. DEPENDENCIAS DE SOFTWARE

Si se desea recompilar el programa, se deberán tener las bibliotecas de la compañía FLI "***libfli-1.99***" (http://www.flicamera.com/software/index.html) y se podrá compilar utilizando los siguientes pasos:

1. g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT fli.o -MD -MP -MF .deps/fli.Tpo -c -o fli.o fli.cpp
2. gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT fli_call.o -MD -MP -MF .deps/fli_call.Tpo -c -o fli_call.o fli_call.c
3. g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT fli_server2.o -MD -MP -MF .deps/fli_server2.Tpo -c -o fli_server2.o fli_server2.cpp
4. g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT server.o -MD -MP -MF .deps/server.Tpo -c -o server.o server.cpp
5. g++ -g -O2 -o fli_server2 fli.o fli_call.o fli_server2.o server.o ./libfli.a -lm -lpthread -static

Este programa fue probado utilizando el sistema operativo Linux Ubuntu 10.04 LTS con el kernel 2.6.32-35. La versión final se ejecuta en la PC dedicada "*SheevaPlug*" con sistema operativo Debian con kernel 2.6.32-5-Kirkwood.

3.4. MÓDULO KERNEL DE LINUX

Para la comunicación vía USB con la cámara, las bibliotecas utilizan el módulo del kernel llamado "**fliusb.ko**" el cual es cargado al iniciar la PC. Este módulo es proporcionado por la compañía FLI. El módulo fue programado para los kernels de Linux 2.6.xx. Si se desea compilar este módulo, se deberá instalar el paquete de los encabezados del kernel de la siguiente manera

sudo apt-get install linux-headers-$(uname -r)

y seguir con las instrucciones que le proporcione la compañía FLI.

3.5. EJECUCIÓN DEL PROGRAMA

No es necesario ninguna instrucción especial o parámetro adicional para correr el programa, simplemente ejecútelo desde una terminal de internet mediante el mando "***./fli_server2***"

4. INSTALACIÓN

En esta sección se describe la instalación de los programas y manejadores *("drivers")* de las cámaras de la compañía FLI.

4.1. INSTALACIÓN DE LOS MANEJADORES (DRIVERS)

Para instalar los "*drivers*" de la cámara en una PC con Linux se deben descargar de http://www.flicamera.com/software/index.html el paquete "**SDk Download**" de Linux el cual se

desempaquetará (fliusb-1.1.tgz, luego se ejecutará el mando "**make**" el cual generará el módulo **fliusb.ko.** Como superusuario, copiar el módulo al directorio "/lib/modules/2.6.32-5-kirkwood/kernel/drivers/usb/"

Para instalar las bibliotecas de la cámara se desempaquetará el archivo libfli-1.99.tgz y se ejecutará el mando "**make**". Como superusuario, ejecutar el mando "**make install**".

4.2. INSTALACIÓN DEL PROGRAMA SERVIDOR FLI_SERVER2

El programa "**fli_server2**" está compilado de manera estática para el procesador ARM de la PC *"SheevaPlug"*, así que sólo es necesario correrlo y estará listo para recibir mandos vía internet en protocolo TCP/IP por el puerto 9710.

5. CONFIGURACIÓN

En esta sección se describirán las diferentes secciones donde hay que configurar nuestro sistema.

5.1. CONFIGURACIÓN DE LA RED

La "*SheevaPlug*" está configurada con una IP estática de la red de instrumentos de los telescopios del OAN-SPM. El archivo con los datos de su configuración se encuentra en "**/etc/network/interface**" y su contenido deberá ser el siguiente:

TABLA 4

Configuración de la red.

```
auto lo
iface lo inet loopback

# The primary network
interface
allow-hotplug etho
auto etho
iface etho inet static
  address 192.168.0.41
  netmask 255.255.255.0
  network 192.168.0.0
  broadcast 192.168.0.255
  gateway 192.168.0.254
```

5.2. SCRIPT DE INICIALIZACIÓN

Al iniciar la *"SheevaPlug",* automáticamente se ejecuta el script de inicialización del sistema operativo "**/etc/rc.local**" en el cual le anexamos la línea "**/root/boot.sh &".** En el archivo *boot.sh*, estarán contenidos todos los procesos de inicialización de nuestro sistema.
La Tabla 5 muestra el contenido de este archivo.
Donde el archivo *"apagame_srv.py*" (ver Apéndice) se utiliza para apagar remotamente la PC de manera segura.

TABLA 5

Archivo de inicialización boot.sh.

```
    echo "configurando fli-
cam"
    modprobe  fliusb2

    #cargar servidor
    cd /root

    #para apagar vía red
    ./apagame_srv.py &

    ./loop.sh
```

El archivo "*loop.sh"* se utiliza para ejecutar el programa servidor.
La Tabla 6 muestra el contenido del archivo *"loop.sh"*.

TABLA 6

Archivo de ciclo del programa servidor.

```
    echo "fli cam server infinite
Loop"
    ./fli_server2
    ./loop.sh
```

6. PRUEBA DEL SISTEMA

Una forma rápida de probar y usar este programa es la siguiente:

1. Ejecutar el programa:
   - **/root/fli-server2 &.**

2. Inicializar la cámara utilizando el siguiente mando:
   - **hose localhost 9710 –out echo "INIT "**.

3. Obtener una imagen de la cámara de un segundo utilizando el siguiente mando:
   - **echo "EXPONE ETIME=100 XSIZE=2048 YSIZE=2048 CBIN=1 RBIN=1 CORG=0 RORG=0 DARK=0 " | hose localhost 9710 --slave**.

4. Pedir la imagen resultante en binario:
   - **echo "MANDABIN " | hose localhost 9710 --slave >imagen.bin**.

Estos mandos se pueden ejecutar desde cualquier PC compatible con el sistema operativo Unix / Linux, las cuales tengan instalado el paquete "**netpipes**". Para esto, cambiar la palabra **localhost** por el IP de la PC en donde se encuentre conectada la cámara, en este caso es 192.168.0.41.

Otro ejemplo, utilizando otra PC como cliente, donde se almacenarían las imágenes y también se desplegarían, sería la siguiente secuencia de mandos:

- **hose 192.168.0.41 9710 –out echo "INIT "**
- **echo "EXPONE ETIME=100 XSIZE=2048 YSIZE=2048 CBIN=1 RBIN=1 CORG=0 RORG=0 DARK=0 " | hose 192.168.0.41 9710 –slave**
- **echo "MANDABIN " | hose 192.168.0.41 9710 --slave >imagen.bin**

y para la visualización de la imagen resultante en la pantalla, utilizaríamos el paquete astronómico *"Saoimage DS9"* de la siguiente manera:

- **cat imagen.bin | xpaset ds9 array [xdim=2048,ydim=2048,zdim=1,bitpix=16]**

7. LA MINI COMPUTADORA "SHEEVAPLUG"

La *Figura 3* muestra varias imágenes de la *"SheevaPlug"*. La *"SheevaPlug"* es una pequeña PC de sólo 110 mm de largo por 69.5 mm de ancho y de un bajo consumo eléctrico (5 Watts) que permite las capacidades de cómputo que un servidor de cómputo en un espacio reducido. La SheevaPlug forma parte de las computadoras tipo "*plug computer*" que son conectadas ("*plug*") directamente a un tomacorriente eléctrico y tiene la apariencia de un adaptador de voltaje CA/CD.

Figura 3: *Imágenes de la PC "SheevaP*lug*"*. Esta PC se conecta directamente a un tomacorriente eléctrico de 115 VAC.

Entre las principales características de cómputo de la "*SheevaPlug*" se encuentran las siguientes:

- Procesador ARM de 1.5 GHz
- Memoria RAM de 512MB DDR2 (400 MHz)
- Memoria NAND Flash de 512MB
- Puerto Ethernet de 100Mhz
- Puerto USB 2.0
- Puerto lector de tarjetas SD
- Reloj de tiempo real con batería.

Debido a su buena capacidad de cómputo para el manejo de la cámara se escogió este modelo, además de su bajo consumo eléctrico y baja disipación de calor.

7.1. SISTEMA OPERATIVO EN TARJETA SD DE 4GB

Para un fácil respaldo y manejo de los programas, el sistema operativo de la *"Sheeva Plug"* se actualizó y se utilizó una tarjeta externa *"SD card*" de una capacidad de 4GB donde contiene el sistema operativo Linux basado en Debian con el kernel 2.6.32-kirkwood armv5tel.

7.2. VARIABLES DE AMBIENTE DE U-BOOT

Al encender la *"SheevaPlug"* ejecutará su programa básico llamado U-BOOT. Éste se encargará de hacer las primeras configuraciones básicas, tal como el programa BIOS los hace en una PC compatible con IBM. Según sus variables definidas, el U-BOOT iniciará desde la memoria interna tipo *"flash*" o desde la memoria externa en la *"SD card"*.

Por omisión, al encender la *"SheevaPlug"* inicializa utilizando la memoria interna. Se modificaron las variables de U-BOOT para que al encender la *"SheevaPlug"* inicie con la memoria externa en la *"SD card"*. La Tabla 7 muestra los valores de las variables del ambiente de U-BOOT.

TABLA 7

Variables de ambiente de U-Boot.

```
baudrate=115200
loads_echo=0
rootpath=/mnt/ARM_FS/
run_diag=yes
console=console=ttyS0,115200
mtdparts=nand_mtd:0xc0000@0(uboot)ro,0x1ff00000@0x100000(root)
CASset=min
MALLOC_len=1
ethprime=egiga0
bootargs_end=:::DB88FXX81:eth0:none
image_name=uImage
standalone=fsload 0x2000000 $(image_name);setenv bootargs $(console)
root=/dev/mtdblock0 rw ip=$(ipaddr):$(serverip)$(bootargs_end) $(mvPhoneConfig); bootm;
ethaddr=00:50:43:9e:25:12
ethmtu=1500
mvPhoneConfig=mv_phone_config=dev0:fxs,dev1:fxs
mvNetConfig=mv_net_config=(00:11:88:0f:62:81,0:1:2:3),mtu=1500
usb0Mode=host
yuk_ethaddr=00:00:00:EE:51:81
nandEcc=1bit
netretry=no
rcvrip=169.254.100.100
loadaddr=0x02000000
autoload=no
ethact=egiga0
arcNumber=2097
boot_nand=nand read.e 0x2000000 0x100000 0x400000
bootargs_console=console=ttyS0,115200
bootargs_root=root=/dev/mmcblk0p2
bootcmd_mmc=mmcinit; ext2load mmc 0:1 0x01100000 /uInitrd; ext2load mmc 0:1
0x00800000 /uImage
bootcmd=setenv bootargs $(bootargs_console) $(bootargs_root); run bootcmd_mmc;
bootm 0x00800000 0x01100000
ipaddr=192.168.0.76
serverip=192.168.0.14
netmask=255.255.255.0
bootargs_tftp=console=ttyS0,115200 ramdisk_size=16384 root=/dev/ram
bootea_tfpt=setenv bootargs $(bootargs_tftp); tftp 0xe00000 uMulti; bootm 0xe00000
stdin=serial
stdout=serial
stderr=serial
mainlineLinux=yes
```

```
enaMonExt=no
enaCpuStream=no
enaWrAllo=no
pexMode=RC
disL2Cache=no
setL2CacheWT=yes
disL2Prefetch=yes
enaICPref=yes
enaDCPref=yes
sata_dma_mode=yes
netbsd_en=no
vxworks_en=no
bootdelay=3
disaMvPnp=no
enaAutoRecovery=yes
pcieTune=no
bootargs=console=ttyS0,115200 root=/dev/mmcblkop2

Environment size: 1598/131068 bytes
```

8. CONCLUSIONES

El programa descrito ha funcionado exitosamente y sin ningún problema desde febrero de 2011 hasta el presente con una cámara FLI de la serie ProLine Modelo PL3041-UV en el telescopio de 0.84 m en el OAN-SPM, en el polarímetro de imagen POLIMA y la rueda de filtros MEXMAN. Además, el convertir el manejo de la cámara de USB a un manejo vía internet, nos permite mayor flexibilidad para utilizar la cámara de una manera robótica o remota.

9. OTROS PROGRAMAS RELACIONADOS

Existe una aplicación gráfica para el usuario, la cual utiliza a este programa y se documentará por separado en el escrito "*Interfaz gráfica para el manejo de los CCD científicos en Python*". Está desarrollado en Python + Gtk.

10. BIBLIOGRAFÍA

- Begining Linux Programming.
  Richard Stones y Neil Matthew.
  Wrox Press.

- FLI SDK Developing Tools.
  http://www.flicamera.com/downloads/FLI_SDK_Documentation.pdf

- Slackware Linux Unleashed
  Bao Ha, Tina Nguyen
  Editorial Sams

- SheevaPlug Computer Forums
  http://www.plugcomputer.org/plugforum/

11. APÉNDICE A

A.1. LISTADO DEL PROGRAMA APAGAME_ SRV.PY.

```
#!/usr/bin/env python
 import os
import socket
import SocketServer
import threading
import thread

#------------------------------------------------------------
class MyTCPHandler(SocketServer.StreamRequestHandler):

  def handle(self):
      data = self.rfile.readline().strip()
      print "llego:",data
      miApp.procesa(data)
      self.request.send('ok \n')
#------------------------------------------------------------
class RUCAFAKE:

 def __init__(self):
      # valores default
      self.mando='/sbin/poweroff'
#------------------------------------------------------------
   #Identifica y procesa los datos recibidos
   def procesa(self,dato):
```

```
        data = str(dato)
        try:
            txt = data.split()
            if txt[0] == "APAGAME":
                print 'clave',txt[1]
                print self.mando
                try:
                        os.system(self.mando)
                except:
                        print "Error"

        except IndexError, e:
            pass
#-----------------------------------------------------------------
if __name__ == "__main__":
    HOST, PORT = "0.0.0.0", 4953
    # Create the server, binding to localhost on port 4967
    SocketServer.allow_reuse_address= True
    server = SocketServer.TCPServer((HOST, PORT), MyTCPHandler)

    miApp = RUCAFAKE()
    print "Escuchando en el puerto", PORT
    server.serve_forever()
```

A.2. REFERENCIA DE SOFTWARE DEL PROGRAMA FLI_SERVER2.

A continuación se listan las referencias de las funciones del programa "***fli_server2***". Esta referencia se realizó con el programa ***"doxygen".***

fli_server2

2.12

Generated by Doxygen 1.6.3

Mon Jul 2 09:18:16 2012

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 cam_t Struct Reference

`#include <fli_call.h>`

Public Attributes

- flidomain_t domain
- char * dname
- char * name

3.1.1 Member Data Documentation

3.1.1.1 char* cam_t::dname

3.1.1.2 flidomain_t cam_t::domain

3.1.1.3 char* cam_t::name

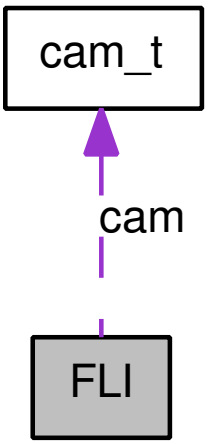
The documentation for this struct was generated from the following file:

- fli_call.h

3.2 FLI Class Reference

`#include <fli.h>`

Collaboration diagram for FLI:



Public Member Functions

- FLI ()
- int expose (int msec, int myfd)
- void estadistica (int *minn, int *maxx, double *mean, double *stdev, int *saturated)
- void showinfo (void)
- double lee_temperatura (void)
- double lee_temperatura_base (void)
- double lee_cooler_power (void)
- void update (void)
- void pon_temperatura (double temp)
- void close (void)
- int init (int myfd)
- void manda_imagen (int myfd)
- int deinterlace (int cols, int rows)
- void tabla (unsigned short *iptr, int cuantos)
- void SetMode (flimode_t mode_index)
- flimode_t GetMode ()
- char * GetMode_String (flimode_t mode_index)
- int init_filters (int myfd)
- void set_filter_pos (int myfd, int pos)
- int get_filter_pos (int myfd)
- ~FLI ()

Public Attributes

- double ccdtemp
- double basetemp

- double power
- bool debug
- bool busy
- bool ccd_init
- bool do_estadistica
- bool dark
- bool full_image
- bool cancela
- int cbin
- int rbin
- int corg
- int rorg
- int etime
- int cols
- int rows
- int imagesize
- int imagePixels
- int maxpixels
- long maxrows
- long maxcols
- long v_corg
- long v_rorg
- long v_cols
- long v_rows
- char model [BUFF_SIZ]
- char error_string [255]
- bool do_deinterlace

3.2.1 Detailed Description

Author

Colorado <colorado@astro.unam.mx>

3.2.2 Constructor & Destructor Documentation

3.2.2.1 FLI::FLI ()

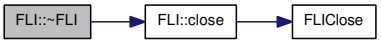
```
64 {
65         cout <<"\nFLI class ready \n";
66         debug=FALSE;
67         ccd_init=FALSE;
68 
69         //maxcols=cols=2154;
70         //maxrows=rows=4608;
71 
72         maxcols=cols=2081;
73         maxrows=rows=2113;
74 
75         maxpixels=maxcols*maxrows;
76         imagesize=cols*rows*2;
77         imagePixels=cols*rows;
78         //shutter=TRUE;
79         etime=100;
80         cbin=1;
```

```
81          rbin=1;
82          rorg=0;
83          corg=0;
84          busy=FALSE;
85          ccdtemp=-10.0;
86          do_estadistica=FALSE;
87          dark=FALSE;
88          cancela=FALSE;
89          //do_deinterlace=true;
90          do_deinterlace=false;
91
92          v_corg=16;
93          v_rorg=32;
94          v_cols=2064;
95          v_rows=2080;
96          is_filter_init=FALSE;
97
98  }
```

3.2.2.2 FLI::~FLI ()

```
101 {
102          cout <<"\nFLI class Ended \n";
103          close();
104 }
```

Here is the call graph for this function:



3.2.3 Member Function Documentation

3.2.3.1 void FLI::close (void)

```
107 {
108
109          cout <<"\nCerrando driver de FLI :"<<cam[0].name<<endl;
110          cout <<"\nccd init :"<<ccd_init<<endl;
111          if ( ccd_init )
112          {
113                  ccd_init=FALSE;
114                  TRYFUNC ( FLIClose, dev );
115                  cout <<"\nCerre dev";
116                  free ( cam[1].name );
117                  cout <<"\nFree Name ";
118                  free ( cam );
119                  cout <<"\nFree Cam \n";
120          }
121 }
```

Here is the call graph for this function:

3.2.3.2 int FLI::deinterlace (int *cols*, int *rows*)

```
457 {
458         unsigned short *old_iptr;
459         unsigned short *new_iptr;
460         int img_size;
461         int half;
462 
463         cout <<"deinterlace cols="<<cols<<" rows=" <<rows<<endl;
464         old_iptr = img;
465         //tabla(old_iptr,10);
466 
467         img_size = maxcols * maxrows * sizeof ( u_int16_t );
468 
469         /* Allocate a new buffer to hold the deinterlaced image. */
470         if ( ( new_iptr = ( u_int16_t * ) malloc ( img_size ) ) == NULL )
471         {
472                 sprintf ( error_string, "Error in allocating memory for deinterla
    cing. \nSaving interlaced image to disk." );
473                 free ( new_iptr );
474                 exit ( 1 );
475                 //return _ERROR;
476         }
477         cout <<"Reserve memoria temporar de "<<img_size<<endl;
478 
479         int i;
480 
481         if ( ( float ) cols/2 != ( int ) cols/2 )
482         {
483                 sprintf ( error_string, "Number of cols must be EVEN for split-se
    rial readout. \nSkipping deinterlace." );
484                 free ( new_iptr );
485                 return _ERROR;
486         }
487 
488         half=cols*rows/2;
489         cout <<"mitad "<<half<<endl;
490         for ( i=0;i< ( cols*rows ) /2;i++ )
491         {
492                 * ( new_iptr+i ) = * ( old_iptr+ ( 2*i ) );
493                 //cout <<i<<"(A) <-- (old)"<<2*i<<endl;
494 
495                 //* ( new_iptr+ ( cols*rows )-i-1 ) = * ( old_iptr+ ( 2*i ) +1 );

496                 //ultiam prueba asi:
497                 * ( new_iptr+ ( half*2 )-i-1 ) = * ( old_iptr+ ( 2*i ) +1 );
498                 //asi estaba
499                 * ( new_iptr+ ( half ) +i+1 ) = * ( old_iptr+ ( 2*i ) +1 );
500                 //cout <<"  (B) "<<half+i+1<< " <-- old="<<       2*i+1 <<endl;
501                 //if (i <20)
502                 {
503                         //cout <<i<<"(A) <-- (old)"<<2*i<<"  (B) "<<half+i+1<< "
    <-- old="<<2*i+1<<endl;
504                 }
505 
506         }
507 
508 
509         //tabla(new_iptr,10);
510         memcpy ( old_iptr, new_iptr, cols*rows*sizeof ( unsigned short ) );
511 
512         free ( new_iptr );
513         return _NO_ERROR;
514 
515 
516 }
```

3.2.3.3 void FLI::estadistica (int * *minn*, int * *maxx*, double * *mean*, double * *stdev*, int * *saturated*)

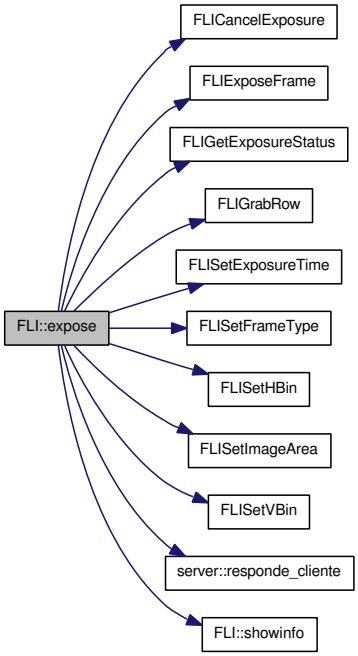
```
402 {
403         unsigned short j;
404         long i;
405         double sum=0.0,sum2=0.0;
406 
407         *saturated=0;
408         for ( i=0;i<imagePixels;i++ )
409         {
410                 j=img[i];
411                 //cout <<i<<".-"<<j<<"   ";
412                 if ( j > *maxx ) *maxx=j;
413                 if ( j < *minn )        *minn=j;
414                 if ( j>= maxushort ) *saturated++;
415 
416                 sum = sum+ ( double ) j;
417                 sum2=sum2+ ( double ) ( j ) * ( double ) ( j );
418         }
419         printf ( "sat %d of %d \n",*saturated,imagePixels );
420 
421         *mean= ( double ) sum/imagePixels;
422         *stdev = pow ( ( sum2 - ( sum*sum ) / ( double ) ( imagePixels ) ) / ( (
      double ) ( imagePixels -1 ) ),0.5 );
423 }
```

3.2.3.4 int FLI::expose (int *msec*, int *myfd*)

```
182 {
183         long tmp1;
184         char manda[100];
185         long nrow;
186 
187         //modificar coordenadas por area visible
188         corg+=v_corg;
189         rorg+=v_rorg;
190         showinfo();
191         TRYFUNC ( FLISetExposureTime, dev, msec );
192 
193         TRYFUNC ( FLISetHBin, dev, cbin );
194 
195         TRYFUNC ( FLISetVBin, dev, rbin );
196 
197         cout <<"cbin="<<cbin<< " rbin=" <<rbin<<endl;
198         cout <<"corg="<<corg<< " cols=" <<cols<<endl;
199         cout <<"rorg="<<rorg<< " rows=" <<rows<<endl;
200         //setting imagen size
201         /*
202         TRYFUNC ( FLISetImageArea, dev, corg, rorg,
203                   (corg +  cols  ) / cbin, (rorg +  rows  ) / rbin );
204 
205 
206 
207         TRYFUNC ( FLISetImageArea, dev, corg*cbin, rorg*rbin,
208                    corg +  (cols  *cbin) ,  rorg +  (rows*rbin)  );
209         */
210         TRYFUNC ( FLISetImageArea, dev, corg, rorg,
211                   corg + ( cols ) ,  rorg + ( rows ) );
212 
213         cout <<"Image upper left corner "<<corg<<" , "<<rorg  <<endl;
214         cout <<"Image lower right corner "<<corg+cols<<" , "<<rorg +  rows   <<en
      dl;
215 
```

```
216 //cout <<"FLISetImageArea "<<corg*cbin<<" "<<rorg*rbin<<" "<< corg +  (cols  *cbi
      n)<<" "<< rorg +  (rows*rbin)<<endl;
217
218         cout <<"FLISetImageArea "<<corg<<" "<<rorg<<" "<< corg + ( cols ) <<" "<<
       rorg + ( rows ) <<endl;
219
220
221
222         //tipo de imagen?
223         if ( dark )
224         {
225                 cout <<"Vamos hacer un dark \n";
226                 TRYFUNC ( FLISetFrameType, dev, FLI_FRAME_TYPE_DARK );
227         }
228         else
229         {
230                 cout <<"Vamos hacer una imagen normal \n";
231                 TRYFUNC ( FLISetFrameType, dev, FLI_FRAME_TYPE_NORMAL );
232         }
233
234         TRYFUNC ( FLIExposeFrame, dev );
235
236         do
237         {
238                 TRYFUNC ( FLIGetExposureStatus, dev, &tmp1 );
239                 if ( r )
240                         break;
241
242                 //usleep ( tmp1 * 1000 );
243                 sprintf ( manda,"TIME_REMAINING %ld \n",tmp1 );
244                 red.responde_cliente ( manda,myfd );
245                 //verificar si hay que cancelar
246                 if ( cancela )
247                 {
248                         //do_cancela (  );
249                         TRYFUNC ( FLICancelExposure, dev );
250                         printf ( "Abortando Expose......!!!!!!!!!! :( ;( \n" );
251                         sprintf ( manda,"CANCELADO \n" );
252                         red.responde_cliente ( manda,myfd );
253                         cancela=FALSE;
254                         return 0;
255                 }
256                 usleep ( 100000 );
257         }
258         while ( tmp1 );
259         printf ( "Ya termine exposicion, voy a leer la imagen del CCD ..........
      ........................\n" );
260         for ( nrow = 0; nrow < rows; nrow++ )
261         {
262                 TRYFUNC ( FLIGrabRow, dev, &img[nrow * cols], cols );
263                 if ( r )
264                         break;
265         }
266
267
268         //grabar imagen
269         //TRYFUNC ( writeraw, ( char * ) "imagen.bin", cols, rows, img );
270         printf ( "Ya termine exposicion ****************************************
      *************************\n" );
271         return !r;
272 }
```

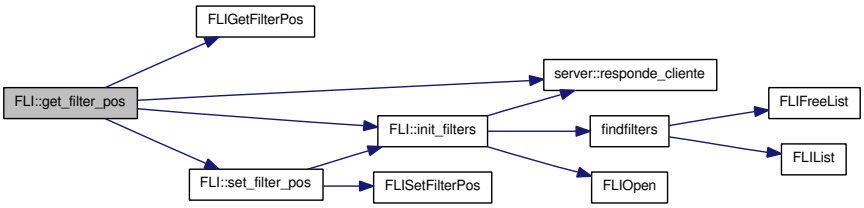
Here is the call graph for this function:



3.2.3.5 int FLI::get_filter_pos (int *myfd*)

```
626 {
627         long pos;
628         char manda[100];
629 
630         if (!is_filter_init){
631 
632                 printf ("Rueda de filtros NO inicializada, voy a Inicializarla..
      \n");
633                 init_filters(myfd);
634                 printf ("Rueda de filtros: Voy a HOME...\n");
635                 set_filter_pos(myfd,0);
636         }
637 
638         printf("Reading filter position  \n");
639         TRYFUNC(FLIGetFilterPos,dev_filter,&pos);
640 
641         sprintf ( manda,"FLI_POS %ld \n",pos );
642         printf(manda);
643         red.responde_cliente ( manda,myfd );
644         return  pos;
645 
646 }
```

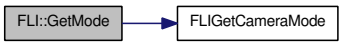
Here is the call graph for this function:



3.2.3.6 flimode_t FLI::GetMode ()

```
540 {
541         static flimode_t mode_index;
542 
543         //FLISetCameraMode(flidev_t dev, flimode_t mode_index)
544         TRYFUNC ( FLIGetCameraMode, dev, &mode_index );
545         return mode_index;
546 }
```

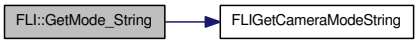
Here is the call graph for this function:



3.2.3.7 char * FLI::GetMode_String (flimode_t *mode_index*)

```
549 {
550         //flimode_t mode_index;
551         static char mode_string[BUFF_SIZ];
552 
553         cout <<"index ="<<mode_index<<endl;
554 
555         TRYFUNC ( FLIGetCameraModeString, dev, mode_index, mode_string,BUFF_SIZ )
    ;
556         cout <<"mode string  ="<<mode_string<<endl;
557 
558         return mode_string;
559 }
```

Here is the call graph for this function:



3.2.3.8 int FLI::init (int *myfd*)

```
275 {
276         int numcams;
277         int i=0,  flushes = 2;
278         long tmp1, tmp2, tmp3, tmp4,  img_rows, row_width;
```

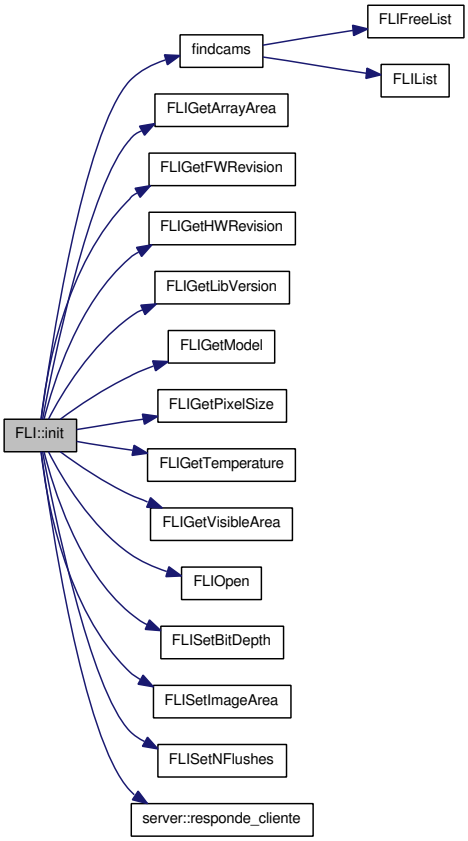
```
279         double d1, d2;
280         char buff[BUFF_SIZ];
281         int img_size;
282 
283         if ( ccd_init==TRUE )
284         {
285                 sprintf ( buff,"info Model: %s\n",model );
286                 red.responde_cliente ( buff,myfd );
287                 sprintf ( buff,"info Visible area: (%ld, %ld)(%ld, %ld) \n",
    v_corg, v_rorg, v_cols, v_rows );
288                 red.responde_cliente ( buff,myfd );
289                 sprintf ( buff,"info Work Size: %d, %d \n", cols,rows );
290                 red.responde_cliente ( buff,myfd );
291                 red.responde_cliente ( ( char * ) "GOOD ->CCD INIT OK \n",myfd );

292                 return 0;
293 
294         }
295         cout << "Inicializando CCD" << endl;
296         //TRYFUNC ( FLISetDebugLevel, NULL /* "NO HOST" */, FLIDEBUG_ALL );
297         //TRYFUNC ( FLISetDebugLevel, NULL /* "NO HOST" */, FLIDEBUG_FAIL );
298 
299         TRYFUNC ( FLIGetLibVersion, libver, LIBVERSIZ );
300         info ( "Library version '%s'", libver );
301         sprintf ( buff,"info Library version: %s\n",libver );
302         red.responde_cliente ( buff,myfd );
303 
304         numcams=findcams ( FLIDOMAIN_USB, &cam );
305         printf ( "cams: %d \n", numcams );
306 
307         if ( numcams<1 )
308         {
309                 //hay problemas
310                 ccd_init=FALSE;
311                 red.responde_cliente ( ( char * ) "FAIL ->Hay problemas con la in
    icializacion \n",myfd );
312                 return -1;
313         }
314 
315         printf ( "\naqui 1\n" );
316 
317         info ( "Trying camera '%s' from %s domain", cam[i].name, cam[i].dname );
318         sprintf ( buff,"info type: %s, -->%s\n",cam[i].name, cam[i].dname );
319         red.responde_cliente ( buff,myfd );
320 
321         TRYFUNC ( FLIOpen, &dev, cam[i].name, FLIDEVICE_CAMERA | cam[i].domain );

322         printf ( "Resultado %ld",r );
323         if ( r<0 )
324         {
325                 //hay problemas
326                 ccd_init=FALSE;
327                 red.responde_cliente ( ( char * ) "FAIL ->Hay problemas con la in
    icializacion \n",myfd );
328                 return -1;
329         }
330         cout << "dev:" << dev <<endl;
331         printf ( "\naqui 2\n" );
332 
333         TRYFUNC ( FLIGetModel, dev, model, BUFF_SIZ );
334         info ( "Model:        %s", model );
335         sprintf ( buff,"info Model: %s\n",model );
336         red.responde_cliente ( buff,myfd );
337 
338         TRYFUNC ( FLIGetHWRevision, dev, &tmp1 );
339         info ( "Hardware Rev: %ld", tmp1 );
340 
```

```
341         TRYFUNC ( FLIGetFWRevision, dev, &tmp1 );
342         info ( "Firmware Rev: %ld", tmp1 );
343 
344         TRYFUNC ( FLIGetPixelSize, dev, &d1, &d2 );
345         info ( "Pixel Size:   %f x %f", d1, d2 );
346         sprintf ( buff,"info pixel size: %f x %f\n",d1, d2 );
347         red.responde_cliente ( buff,myfd );
348 
349         TRYFUNC ( FLIGetArrayArea, dev, &tmp1, &tmp2, &maxcols, &maxrows );
350         info ( "Array area:   (%ld, %ld)(%ld, %ld)", tmp1, tmp2, maxcols,
    maxrows );
351         sprintf ( buff,"info Array area: (%ld, %ld)(%ld, %ld) \n", tmp1, tmp2,
    maxcols, maxrows );
352         red.responde_cliente ( buff,myfd );
353 
354         TRYFUNC ( FLIGetVisibleArea, dev, &v_corg, &v_rorg, &v_cols, &v_rows );
355 
356         info ( "Visible area: (%ld, %ld)(%ld, %ld)", v_corg, v_rorg, v_cols,
    v_rows );
357 
358         cols=row_width = ( v_cols - v_corg ) / cbin;
359         rows=img_rows = ( v_rows - v_rorg ) / rbin;
360         sprintf ( buff,"info Visible area: (%ld, %ld)(%ld, %ld) \n", v_corg,
    v_rorg, v_cols, v_rows );
361         red.responde_cliente ( buff,myfd );
362         sprintf ( buff,"info Work Size: %d, %d \n", cols,rows );
363         red.responde_cliente ( buff,myfd );
364 
365         //poner modo 16 bits, algunas camaras no lo soportan
366         TRYFUNC ( FLISetBitDepth, dev, FLI_MODE_16BIT );
367 
368         tmp1=v_corg;
369         tmp2=v_rorg;
370         tmp3=v_cols;
371         tmp4=v_rows;
372         cout <<"FLISetImageArea "<<tmp1<<" "<<tmp2<<" "<<tmp1 + ( tmp3 - tmp1 ) <
    <" "<<tmp2 + ( tmp4 - tmp2 ) / rbin<<endl;
373         TRYFUNC ( FLISetImageArea, dev, tmp1, tmp2,
374                   tmp1 + ( tmp3 - tmp1 ) / cbin, tmp2 + ( tmp4 - tmp2 ) / rbin );
375 
376         TRYFUNC ( FLISetNFlushes, dev, flushes );
377 
378         TRYFUNC ( FLIGetTemperature, dev, &ccdtemp );
379         info ( "Temperature:  %f", ccdtemp );
380         sprintf ( buff,"LEE_TEMP %2.1f\n",ccdtemp );
381         red.responde_cliente ( buff,myfd );
382 
383         //img_size = img_rows * row_width * sizeof ( u_int16_t );
384         img_size = maxcols * maxrows * sizeof ( u_int16_t );
385         printf ( "Voy a reservar memoria de  %d Bytes \n", img_size );
386 
387         if ( ( img = ( u_int16_t * ) malloc ( img_size ) ) == NULL )
388         {
389                 err ( 1, "malloc() failed" );
390                 exit ( 1 );
391         }
392 
393 
394         printf ( "Image buffer: 0x%X ( %d ) \n", img, img );
395 
396         ccd_init=TRUE;
397         red.responde_cliente ( ( char * ) "GOOD ->CCD INIT OK \n",myfd );
398         return 1;
399 }
```

Here is the call graph for this function:

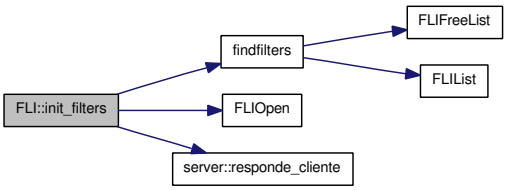


3.2.3.9 int FLI::init_filters (int *myfd*)

```
562 {
563         int numcams;
564         int i=0,  flushes = 2;
565         long tmp1, tmp2, tmp3, tmp4,  img_rows, row_width;
566         double d1, d2;
567         char buff[BUFF_SIZ];
568         int img_size;
569 
570 
571         cout << "Inicializando Rueda de filtros" << endl;
572         //TRYFUNC ( FLISetDebugLevel, NULL /* "NO HOST" */, FLIDEBUG_ALL );
573         //TRYFUNC ( FLISetDebugLevel, NULL /* "NO HOST" */, FLIDEBUG_FAIL );
574 
575 
576 
577         numcams=findfilters ( FLIDOMAIN_USB, &cam );
578         printf ( "cams: %d \n", numcams );
579 
580         if ( numcams<1 )
581         {
582                 //hay problemas
583                 ccd_init=FALSE;
584                 red.responde_cliente ( ( char * ) "FAIL ->Hay problemas con la in
      icializacion de la rueda de filtros \n",myfd );
585                 return -1;
```

```
586         }
587
588         printf ( "\naqui 1\n" );
589
590         info ( "Trying filter wheels '%s' from %s domain", cam[i].name, cam[i].dn
    ame );
591         sprintf ( buff,"info type: %s, -->%s\n",cam[i].name, cam[i].dname );
592         red.responde_cliente ( buff,myfd );
593
594         TRYFUNC ( FLIOpen, &dev_filter, cam[i].name, FLIDEVICE_FILTERWHEEL | cam[
    i].domain );
595         printf ( "Resultado %ld",r );
596         if ( r<0 )
597         {
598                 //hay problemas
599                 ccd_init=FALSE;
600                 red.responde_cliente ( ( char * ) "FAIL ->Hay problemas con la in
    icializacion de la rueda de filtros\n",myfd );
601                 return -1;
602         }
603         cout << "dev:" << dev_filter <<endl;
604         printf ( "\naqui 2\n" );
605         is_filter_init=TRUE;
606
607         return 1;
608 }
```

Here is the call graph for this function:



3.2.3.10 double FLI::lee_cooler_power (void)

```
153 {
154         TRYFUNC ( FLIGetCoolerPower  ,dev, &power);
155         return power;
156 }
```

Here is the call graph for this function:

3.2.3.11 double FLI::lee_temperatura (void)

```
140 {
141         TRYFUNC ( FLIGetTemperature, dev, &ccdtemp );
142         return ccdtemp;
143 }
```

Here is the call graph for this function:

3.2.3.12 double FLI::lee_temperatura_base (void)

```
146 {
147         //TRYFUNC ( FLIGetTemperature, dev, &ccdtemp );
148         TRYFUNC ( FLIReadTemperature ,dev, FLI_TEMPERATURE_BASE, &basetemp);
149         return basetemp;
150 }
```

Here is the call graph for this function:

3.2.3.13 void FLI::manda_imagen (int *myfd*)

```
427 {
428         int i;
429         int tx=0;
430         unsigned short pixel;
431         //char manda[100];
432         FILE *fnet;
433 
434         cout <<"En mandabin con descriptor "<<myfd<<endl;
435 
436         fnet = fdopen ( dup ( myfd ), "w" );
437         if ( ( fnet == NULL ) )
438         {
439                 cout <<" Err en el fdopen fnet\n";
440         }
441 
442         cout <<"Voy a madar imagen via red "<<imagePixels<<" Fd="<<myfd<<endl;
443 
444         tx=0;
445         for ( i=0;i<imagePixels;i++ )
446         {
447                 pixel=img[i];
448                 tx+=fwrite ( &pixel,sizeof ( pixel ),1,fnet );
449         }
450         cout <<i<<" Mande = "<<pixel<<" bytes, actual de="<<tx<<" (Total="<<image
      Pixels<<" )"<<endl;
451         fflush ( fnet );
452         fclose ( fnet );
453 
454 }
```

3.2.3.14 void FLI::pon_temperatura (double *temp*)

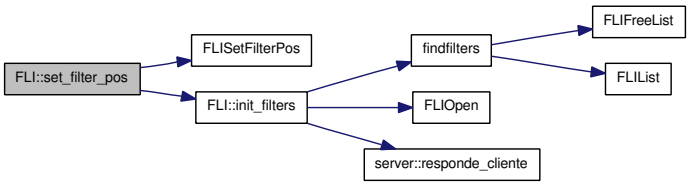
```
160 {
161         cout << "Regular temperatura a:" << temp <<endl;
162         TRYFUNC ( FLISetTemperature, dev, temp );
163         cout << "dev:" << dev <<endl;
164 }
```

Here is the call graph for this function:

3.2.3.15 void FLI::set_filter_pos (int *myfd*, int *pos*)

```
613 {
614         if (!is_filter_init){
615 
616                 printf ("Rueda de filtros NO inicializada, voy a Inicializarla..
      \n");
617                 init_filters(myfd);
618         }
619 
620         printf("Moving filter to: %d \n",pos);
621         TRYFUNC(FLISetFilterPos,dev_filter,pos);
622         cout << "dev:" << dev_filter <<endl;
623 }
```

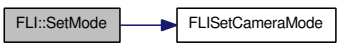
Here is the call graph for this function:



3.2.3.16 void FLI::SetMode (flimode_t *mode_index*)

```
531 {
532 
533         //FLISetCameraMode(flidev_t dev, flimode_t mode_index)
534         TRYFUNC ( FLISetCameraMode, dev, mode_index );
535 }
```

Here is the call graph for this function:



3.2.3.17 void FLI::showinfo (void)

```
167 {
168         printf ( "\n Showinfo:" );
169         printf ( "\n Modelo=%s",model );
170         printf ( "\n ETIME=%d",etime );
171         printf ( "\n XSIZE=%d",cols );
172         printf ( "\n YSIZE=%d",rows );
173         printf ( "\n CBIN=%d",cbin );
```

```
174         printf ( "\n RBIN=%d",rbin );
175         printf ( "\n CORG=%d",corg );
176         printf ( "\n RORG=%d",rorg );
177         printf ( "\n DARK=%d",dark );
178         printf ( "\n DEBUG=%d\n",debug );
179 }
```

3.2.3.18 void FLI::tabla (unsigned short * *iptr*, int *cuantos*)

```
519 {
520         unsigned short pixel;
521         int i;
522
523         for ( i=0;i<cuantos;i++ )
524         {
525                 pixel=iptr[i];
526                 cout <<i<<" = "<<pixel<<endl;
527         }
528 }
```

3.2.3.19 void FLI::update (void)

```
124 {
125         int x,y;
126
127         x=maxcols/cbin;
128         y=maxrows/rbin;
129
130         // revisa que el numero de columnas y renglones sea par
131         if ( x % 2 != 0 ) x--; //quitarle un numero para hacerlo par
132         if ( y % 2 != 0 ) y--; //quitarle un numero para hacerlo par
133
134         maxpixels=x*y;
135         imagesize=cols*rows*2;
136         imagePixels=cols*rows;
137 }
```

3.2.4 Member Data Documentation

3.2.4.1 double FLI::basetemp

3.2.4.2 bool FLI::busy

3.2.4.3 bool FLI::cancela

3.2.4.4 int FLI::cbin

3.2.4.5 bool FLI::ccd_init

3.2.4.6 double FLI::ccdtemp

3.2.4.7 int FLI::cols

3.2.4.8 int FLI::corg

3.2.4.9 bool FLI::dark

3.2.4.10 bool FLI::debug

3.2.4.11 bool FLI::do_deinterlace

3.2.4.12 bool FLI::do_estadistica

3.2.4.13 char FLI::error_string[255]

3.2.4.14 int FLI::etime

3.2.4.15 bool FLI::full_image

3.2.4.16 int FLI::imagePixels

3.2.4.17 int FLI::imagesize

3.2.4.18 long FLI::maxcols

3.2.4.19 int FLI::maxpixels

3.2.4.20 long FLI::maxrows

3.2.4.21 char FLI::model[BUFF_SIZ]

3.2.4.22 double FLI::power

3.2.4.23 int FLI::rbin

3.2.4.24 int FLI::rorg

3.2.4.25 int FLI::rows

3.2.4.26 long FLI::v_cols

3.2.4.27 long FLI::v_corg

3.2.4.28 long FLI::v_rorg

3.2.4.29 long FLI::v_rows

The documentation for this class was generated from the following files:

- fli.h
- fli.cpp

3.3 micliente Struct Reference

`#include <server.h>`

Public Attributes

- string instruccion
- int local_fd
- char ip [20]

3.3.1 Member Data Documentation

3.3.1.1 string micliente::instruccion

3.3.1.2 char micliente::ip[20]

3.3.1.3 int micliente::local_fd

The documentation for this struct was generated from the following file:

- server.h

3.4 server Class Reference

`#include <server.h>`

Public Member Functions

- server ()
- ∼server ()
- void init (int myport)
- void netclose (void)
- void set_ip (char ∗newip)
- void revisa ()
- int checa ()
- void responde_cliente (char ∗mensaje, int myfd)

Public Attributes

- short op
- char ip [20]
- short procesando
- bool salida
- bool busy
- short status
- short debug
- char instruccion [200]
- bool CANCELA
- int client_sockfd
- FILE ∗ fileSal

3.4.1 Constructor & Destructor Documentation

3.4.1.1 server::server ()

```
34 {
35         cout <<"Net Server V1.2 loaded..."<<endl;
36         ch=1;
37         CANCELA=false;
38         salida=false;
39 }
```

3.4.1.2 server::∼server ()

```
42 {
43         cout <<"Closing server side..."<<endl;
44 }
```

3.4.2 Member Function Documentation

3.4.2.1 int server::checa ()

```
85 {
86 //      int i;
87         debug=1;
88
89         if ( debug ) cout <<"Esperando datos de red\n"<<endl;
90         client_len=sizeof ( client_address );
91         //esperar una nueva conexion
92         client_sockfd=accept ( server_sockfd, ( struct sockaddr * ) &client_addre
     ss, ( size_t * ) &client_len );
93
94         //cout <<"FD del cliente "<<client_sockfd<<endl;
95         //if ( client_sockfd<0 )
96         return client_sockfd;
97 /*
98         set_ip ( inet_ntoa ( client_address.sin_addr ) ); // return the IP
99
100         if ( debug >1 ) printf ( "\nadding client on fd %d \n",client_sockfd );
101         i=read ( client_sockfd,&instruccion,sizeof ( instruccion ) );
102
103         return ( i );
104 */
105 } //end red_revisa
```

3.4.2.2 void server::init (int *myport*)

```
48 {
49         int e;
50         int flag = 1;
51
52         if ( debug ) printf ( "\nServer init...\n" );
53         //crea socket para el servidor sin nombre
54         server_sockfd=socket ( AF_INET,SOCK_STREAM,0 );
55         //define las caracteristicas del socket
56         server_address.sin_family=AF_INET;
57         server_address.sin_addr.s_addr=htonl ( INADDR_ANY );
58         server_address.sin_port=htons ( myport );
59         memset ( server_address.sin_zero, '\0', sizeof server_address.sin_zero );

60
61         //fijar opciones del socket
62         e=setsockopt ( server_sockfd, SOL_SOCKET, SO_REUSEADDR,&flag, sizeof ( fl
     ag ) );
63
64         if ( debug ) printf ( "\nsetsockopt=%d",e );
65         if ( e<0 ) perror ( "Error:" );
66
67         //asignar el nombre al socket, fijar mi ip y puerto
68         server_len=sizeof ( server_address );
69         e=bind ( server_sockfd, ( struct sockaddr * ) &server_address,server_len
     );
70         if ( debug ) printf ( "\nbind=%d",e );
71         if ( e<0 )
72         {
73                 perror ( "Error: Al conectarse por via red BIND:" );
74                 printf ( "\nEsperar un minuto antes de volver a correr el program
     a\n\n" );
75                 exit ( 1 );
76         }
77         //crea queue para 5 conexiones
78         e=listen ( server_sockfd,5 );
```

```
79         if ( debug ) printf ( "\nlisten=%d",e );
80         if ( debug ) printf ( "\nserver init...END \n" );
81 }
```

3.4.2.3 void server::netclose (void)

```
108 {
109         if ( debug ) printf ( "\nCerrando conexiones de red del servidor" );
110         close ( server_sockfd );
111         if ( debug ) printf ( "\nYA CERRE TODAS LA CONEXIONES red\n" );
112 }
```

3.4.2.4 void server::responde_cliente (char * *mensaje*, int *myfd*)

```
122 {
123         static int i=0;
124         //FILE *fnet;
125         char salida[200];
126         int j;
127
128         strcpy ( salida,mensaje );
129
130 /*
131         fnet = fdopen ( dup ( myfd ), "w" );
132         if ( ( fnet == NULL ) )
133         {
134                 cout <<" Err en el fdopen fnet casi_expone\n";
135         }
136
137         fwrite ( salida,strlen ( salida ),1,fnet );
138         //fflush(fnet);
139         fclose ( fnet );
140         i++;
141         cout <<"mande a cliente ("<<myfd<<") : "<<salida<<" #="<<i<<endl;
142 */
143         i++;
144         j=write(myfd,salida,strlen ( salida ));
145         cout <<"mande a cliente ("<<myfd<<") : "<<salida<<" #="<<j<<endl;
146 }
```

3.4.2.5 void server::revisa ()

3.4.2.6 void server::set_ip (char * *newip*)

```
115 {
116         //if (debug) printf("old ip %s",ip);
117         strcpy ( ip,newip );
118         //if (debug) printf("new ip %s \n",newip);
119 }
```

3.4.3 Member Data Documentation

3.4.3.1 bool server::busy

3.4.3.2 bool server::CANCELA

3.4.3.3 int server::client_sockfd

3.4.3.4 short server::debug

3.4.3.5 FILE* server::fileSal

3.4.3.6 char server::instruccion[200]

3.4.3.7 char server::ip[20]

3.4.3.8 short server::op

3.4.3.9 short server::procesando

3.4.3.10 bool server::salida

3.4.3.11 short server::status

The documentation for this class was generated from the following files:

- server.h
- server.cpp

Chapter 4

File Documentation

4.1 fli.cpp File Reference

```
#include "fli.h"
#include <sys/types.h>
#include <iostream>
#include <cstdlib>
#include <math.h>
#include <sys/time.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string>
#include <sys/stat.h>
#include <stdarg.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <err.h>
#include <limits.h>
#include <errno.h>
```

Include dependency graph for fli.cpp:

This graph shows which files directly or indirectly include this file:

Defines

- #define TRYFUNC(f,...)
- #define info(format, args...) printf("%s: " format "\n", __progname, ## args)
- #define warnc(c, format, args...) warnx(format ": %s", ## args, strerror(c))
- #define LIBVERSIZ 1024

Functions

- int findcams (flidomain_t domain, cam_t **cam)
- int writeraw (char *filename, int width, int height, void *data)
- int findfilters (flidomain_t domain, cam_t **cam)

Variables

- const unsigned short maxushort = 65534
- const char * __progname
- server red
- pthread_mutex_t mutex_busy

4.1.1 Define Documentation

4.1.1.1 #define info(format, args...) printf("%s: " format "\n", __progname, ## args)

4.1.1.2 #define LIBVERSIZ 1024

4.1.1.3 #define TRYFUNC(f, ...)

Value:

```
do {                                            \
    if ((r = f(__VA_ARGS__)))                   \
      warnc(-r, #f "() failed");                \
  } while (0)
```

4.1.1.4 #define warnc(c, format, args...) warnx(format ": %s", ## args, strerror(c))

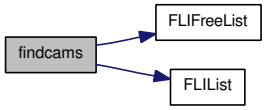
4.1.2 Function Documentation

4.1.2.1 int findcams (flidomain_t *domain*, cam_t ** *cam*)

```
25 {
26         long r;
27         char **tmplist;
```

```
28          int numcams = 0;
29
30          printf ( "\nfinding cams...\n" );
31
32          TRYFUNC ( FLIList, domain | FLIDEVICE_CAMERA, &tmplist );
33
34          if ( tmplist != NULL && tmplist[0] != NULL )
35          {
36                  int i, cams = 0;
37
38                  for ( i = 0; tmplist[i] != NULL; i++ )
39                          cams++;
40
41                  if ( ( *cam = (cam_t*)realloc ( *cam, ( numcams + cams ) * sizeof
        ( cam_t ) ) ) == NULL )
42                          err ( 1, "realloc() failed" );
43
44                  for ( i = 0; tmplist[i] != NULL; i++ )
45                  {
46                          int j;
47                          cam_t *tmpcam = *cam + i;
48
49                          for ( j = 0; tmplist[i][j] != '\0'; j++ )
50                                  if ( tmplist[i][j] == ';' )
51                                  {
52                                          tmplist[i][j] = '\0';
53                                          break;
54                                  }
55
56                          tmpcam->domain = domain;
57                          switch ( domain )
58                          {
59                                  case FLIDOMAIN_PARALLEL_PORT:
60                                          tmpcam->dname = "parallel port";
61                                          break;
62
63                                  case FLIDOMAIN_USB:
64                                          tmpcam->dname = "USB";
65                                          break;
66
67                                  case FLIDOMAIN_SERIAL:
68                                          tmpcam->dname = "serial";
69                                          break;
70
71                                  case FLIDOMAIN_INET:
72                                          tmpcam->dname = "inet";
73                                          break;
74
75                                  default:
76                                          tmpcam->dname = "Unknown domain";
77                                          break;
78                          }
79                          tmpcam->name = strdup ( tmplist[i] );
80                  }
81
82                  numcams += cams;
83          }
84
85          TRYFUNC ( FLIFreeList, tmplist );
86
87          printf ( "Encontre %d camaras \n",numcams );
88          return numcams;
89  }
```

Here is the call graph for this function:

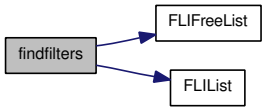


4.1.2.2 int findfilters (flidomain_t *domain*, cam_t ** *cam*)

```
118 {
119         long r;
120         char **tmplist;
121         int numcams = 0;
122 
123         printf ( "\nfinding filter wheels\n" );
124 
125         TRYFUNC ( FLIList, domain | FLIDEVICE_FILTERWHEEL, &tmplist );
126         printf("list %d \n",tmplist);
127         printf("list %d \n",tmplist[0]);
128 
129         if ( tmplist != NULL && tmplist[0] != NULL )
130         {
131                 int i, cams = 0;
132 
133                 for ( i = 0; tmplist[i] != NULL; i++ )
134                         cams++;
135 
136                 if ( ( *cam = (cam_t*)realloc ( *cam, ( numcams + cams ) * sizeof
      ( cam_t ) ) ) == NULL )
137                         err ( 1, "realloc() failed" );
138 
139                 for ( i = 0; tmplist[i] != NULL; i++ )
140                 {
141                         int j;
142                         cam_t *tmpcam = *cam + i;
143 
144                         for ( j = 0; tmplist[i][j] != '\0'; j++ )
145                                 if ( tmplist[i][j] == ';' )
146                                 {
147                                         tmplist[i][j] = '\0';
148                                         break;
149                                 }
150 
151                         tmpcam->domain = domain;
152                         switch ( domain )
153                         {
154                                 case FLIDOMAIN_PARALLEL_PORT:
155                                         tmpcam->dname = "parallel port";
156                                         break;
157 
158                                 case FLIDOMAIN_USB:
159                                         tmpcam->dname = "USB";
160                                         break;
161 
162                                 case FLIDOMAIN_SERIAL:
163                                         tmpcam->dname = "serial";
164                                         break;
165 
166                                 case FLIDOMAIN_INET:
167                                         tmpcam->dname = "inet";
168                                         break;
169 
170                                 default:
```

```
171                                         tmpcam->dname = "Unknown domain";
172                                         break;
173                         }
174                         tmpcam->name = strdup ( tmplist[i] );
175                 }
176
177                 numcams += cams;
178         }  else {
179                 printf ( "No Encontre NADA  \n" );
180         }
181
182
183         TRYFUNC ( FLIFreeList, tmplist );
184
185         printf ( "Encontre %d camaras \n",numcams );
186         return numcams;
187 }
```

Here is the call graph for this function:



4.1.2.3 int writeraw (char * *filename*, int *width*, int *height*, void * *data*)

```
92 {
93         int fd, size, err;
94
95         if ( ( fd = open ( filename, O_WRONLY | O_CREAT | /* O_EXCL */ O_TRUNC,
96                            S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH ) ) == -1 )
97         {
98                 warn ( "open(%s) failed", filename );
99                 return -errno;
100         }
101
102         size = width * height * sizeof ( u_int16_t );
103         if ( ( err = write ( fd, data, size ) ) != size )
104         {
105                 warn ( "write() failed" );
106                 err = -errno;
107         }
108         else
109                 err = 0;
110
111         close ( fd );
112
113         return err;
114 }
```

4.1.3 Variable Documentation

4.1.3.1 const char* __progname

4.1.3.2 const unsigned short maxushort = 65534

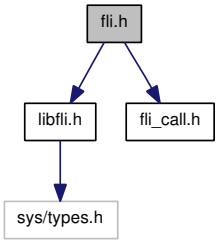
4.1.3.3 pthread_mutex_t mutex_busy

4.1.3.4 server red

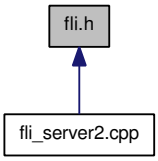
4.2 fli.h File Reference

```
#include "libfli.h"
#include "fli_call.h"
```

Include dependency graph for fli.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FLI

Defines

- #define TRUE 1
- #define FALSE 0
- #define BUFF_SIZ 4096
- #define _ERROR -1
- #define _NO_ERROR 0

4.2.1 Define Documentation

4.2.1.1 #define _ERROR -1

4.2.1.2 #define _NO_ERROR 0

4.2.1.3 #define BUFF_SIZ 4096

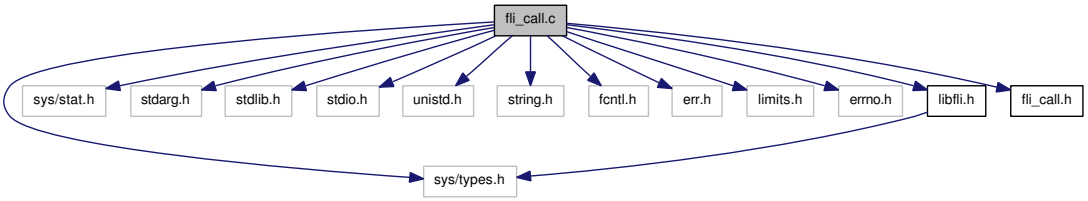
4.2.1.4 #define FALSE 0

4.2.1.5 #define TRUE 1

4.3 fli_call.c File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <err.h>
#include <limits.h>
#include <errno.h>
#include "libfli.h"
#include "fli_call.h"
```

Include dependency graph for fli_call.c:



Functions

- int findcams (flidomain_t domain, cam_t **cam)
- int findfilters (flidomain_t domain, cam_t **cam)
- int writeraw (char *filename, int width, int height, void *data)

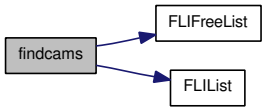
4.3.1 Function Documentation

4.3.1.1 int findcams (flidomain_t *domain*, cam_t ** *cam*)

```
25 {
26         long r;
27         char **tmplist;
28         int numcams = 0;
29 
30         printf ( "\nfinding cams...\n" );
31 
32         TRYFUNC ( FLIList, domain | FLIDEVICE_CAMERA, &tmplist );
33 
34         if ( tmplist != NULL && tmplist[0] != NULL )
35         {
36                 int i, cams = 0;
37 
```

```
38                 for ( i = 0; tmplist[i] != NULL; i++ )
39                         cams++;
40
41                 if ( ( *cam = (cam_t*)realloc ( *cam, ( numcams + cams ) * sizeof
       ( cam_t ) ) ) == NULL )
42                         err ( 1, "realloc() failed" );
43
44                 for ( i = 0; tmplist[i] != NULL; i++ )
45                 {
46                         int j;
47                         cam_t *tmpcam = *cam + i;
48
49                         for ( j = 0; tmplist[i][j] != '\0'; j++ )
50                                 if ( tmplist[i][j] == ';' )
51                                 {
52                                         tmplist[i][j] = '\0';
53                                         break;
54                                 }
55
56                         tmpcam->domain = domain;
57                         switch ( domain )
58                         {
59                                 case FLIDOMAIN_PARALLEL_PORT:
60                                         tmpcam->dname = "parallel port";
61                                         break;
62
63                                 case FLIDOMAIN_USB:
64                                         tmpcam->dname = "USB";
65                                         break;
66
67                                 case FLIDOMAIN_SERIAL:
68                                         tmpcam->dname = "serial";
69                                         break;
70
71                                 case FLIDOMAIN_INET:
72                                         tmpcam->dname = "inet";
73                                         break;
74
75                                 default:
76                                         tmpcam->dname = "Unknown domain";
77                                         break;
78                         }
79                         tmpcam->name = strdup ( tmplist[i] );
80                 }
81
82                 numcams += cams;
83         }
84
85         TRYFUNC ( FLIFreeList, tmplist );
86
87         printf ( "Encontre %d camaras \n",numcams );
88         return numcams;
89 }
```

Here is the call graph for this function:

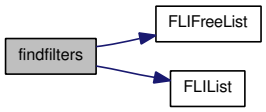


4.3.1.2 int findfilters (flidomain_t *domain*, cam_t ** *cam*)

```
118 {
119         long r;
120         char **tmplist;
121         int numcams = 0;
122 
123         printf ( "\nfinding filter wheels\n" );
124 
125         TRYFUNC ( FLIList, domain | FLIDEVICE_FILTERWHEEL, &tmplist );
126         printf("list %d \n",tmplist);
127         printf("list %d \n",tmplist[0]);
128 
129         if ( tmplist != NULL && tmplist[0] != NULL )
130         {
131                 int i, cams = 0;
132 
133                 for ( i = 0; tmplist[i] != NULL; i++ )
134                         cams++;
135 
136                 if ( ( *cam = (cam_t*)realloc ( *cam, ( numcams + cams ) * sizeof
      ( cam_t ) ) ) == NULL )
137                         err ( 1, "realloc() failed" );
138 
139                 for ( i = 0; tmplist[i] != NULL; i++ )
140                 {
141                         int j;
142                         cam_t *tmpcam = *cam + i;
143 
144                         for ( j = 0; tmplist[i][j] != '\0'; j++ )
145                                 if ( tmplist[i][j] == ';' )
146                                 {
147                                         tmplist[i][j] = '\0';
148                                         break;
149                                 }
150 
151                         tmpcam->domain = domain;
152                         switch ( domain )
153                         {
154                                 case FLIDOMAIN_PARALLEL_PORT:
155                                         tmpcam->dname = "parallel port";
156                                         break;
157 
158                                 case FLIDOMAIN_USB:
159                                         tmpcam->dname = "USB";
160                                         break;
161 
162                                 case FLIDOMAIN_SERIAL:
163                                         tmpcam->dname = "serial";
164                                         break;
165 
166                                 case FLIDOMAIN_INET:
167                                         tmpcam->dname = "inet";
168                                         break;
169 
170                                 default:
171                                         tmpcam->dname = "Unknown domain";
172                                         break;
173                         }
174                         tmpcam->name = strdup ( tmplist[i] );
175                 }
176 
177                 numcams += cams;
178         }  else {
179                 printf ( "No Encontre NADA  \n" );
180         }
181 
```

```
182
183         TRYFUNC ( FLIFreeList, tmplist );
184
185         printf ( "Encontre %d camaras \n",numcams );
186         return numcams;
187 }
```

Here is the call graph for this function:

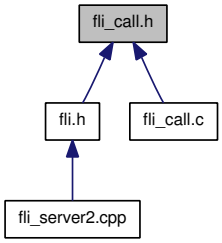


4.3.1.3 int writeraw (char * *filename*, int *width*, int *height*, void * *data*)

```
92 {
93         int fd, size, err;
94
95         if ( ( fd = open ( filename, O_WRONLY | O_CREAT | /* O_EXCL */ O_TRUNC,
96                            S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH ) ) == -1 )
97         {
98                 warn ( "open(%s) failed", filename );
99                 return -errno;
100         }
101
102         size = width * height * sizeof ( u_int16_t );
103         if ( ( err = write ( fd, data, size ) ) != size )
104         {
105                 warn ( "write() failed" );
106                 err = -errno;
107         }
108         else
109                 err = 0;
110
111         close ( fd );
112
113         return err;
114 }
```

4.4 fli_call.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct cam_t

Defines

- #define TRYFUNC(f,...)
- #define info(format, args...) printf("%s: " format "\n", __progname, ## args)
- #define warnc(c, format, args...) warnx(format ": %s", ## args, strerror(c))
- #define LIBVERSIZ 1024

4.4.1 Define Documentation

4.4.1.1 #define info(format, args...) printf("%s: " format "\n", __progname, ## args)

4.4.1.2 #define LIBVERSIZ 1024

4.4.1.3 #define TRYFUNC(f, ...)

Value:

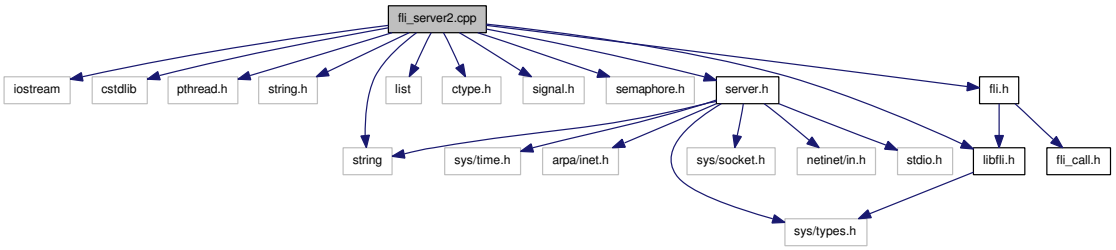
```
do {                                            \
    if ((r = f(__VA_ARGS__)))                   \
      warnc(-r, #f "() failed");                \
  } while (0)
```

4.4.1.4 #define warnc(c, format, args...) warnx(format ": %s", ## args, strerror(c))

4.5 fli_server2.cpp File Reference

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <string.h>
#include <string>
#include <list>
#include <ctype.h>
#include <signal.h>
#include <semaphore.h>
#include "server.h"
#include "fli.h"
#include "libfli.h"
```

Include dependency graph for fli_server2.cpp:



Defines

- #define PORT 9710
- #define MyVersion "2.12"
- #define TRUE 1
- #define FALSE 0

Functions

- void * onthread_procesa_mando (void *arg)
- int procesa_mando (int myfd, char *txt)
- void casi_expone (char *params, int myfd)
- void test ()
- int main (int argc, char *argv[ ])

Variables

- server red
- pthread_t thread_red

- pthread_mutex_t mutex_busy
- void * t_result
- int res
- pthread_t thread_mando
- list< micliente > lista
- list< micliente >::iterator i
- FLI ccd

4.5.1 Define Documentation

4.5.1.1 #define FALSE 0

4.5.1.2 #define MyVersion "2.12"

4.5.1.3 #define PORT 9710

4.5.1.4 #define TRUE 1

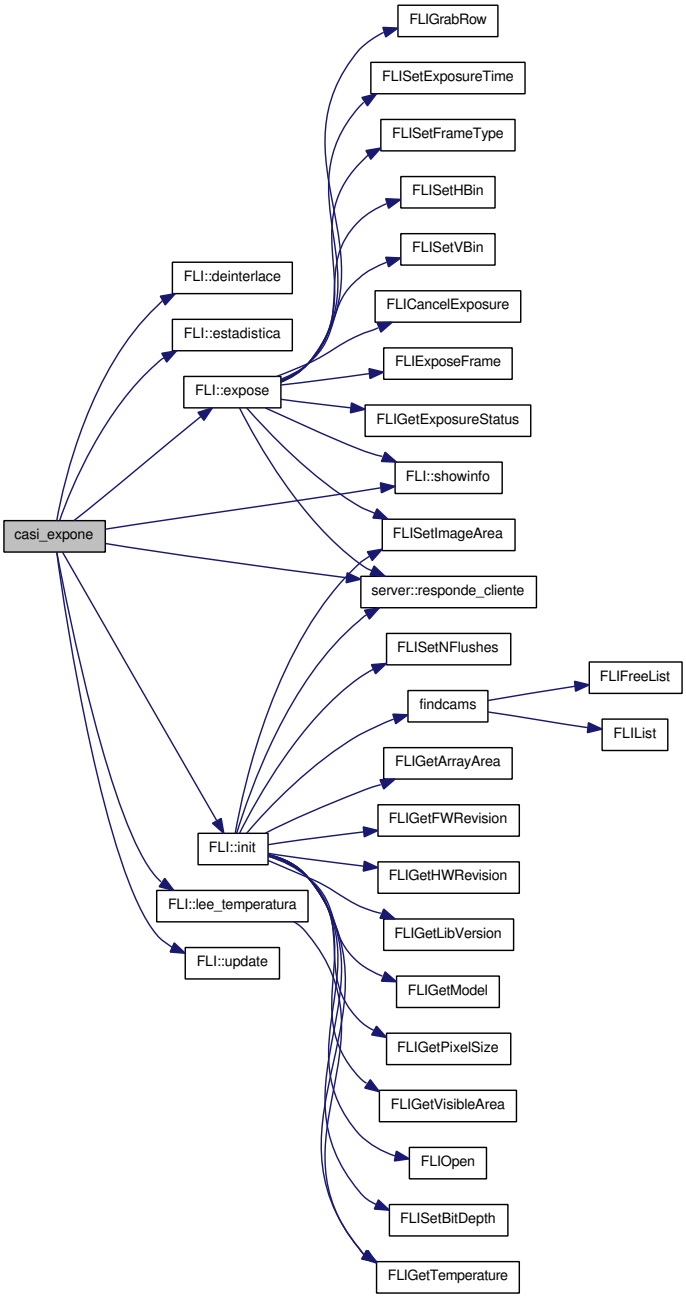
4.5.2 Function Documentation

4.5.2.1 void casi_expone (char * *params*, int *myfd*)

```
426 {
427         float t;
428         char mensaje[200];
429         int rx;
430         int minn=70000, maxx=0,saturated=0;
431         double mean,stdev;
432         int error;
433 
434         strcpy ( mensaje,params );
435         cout <<"En casi expone con  params = "<<mensaje<<endl;
436 
437 
438 //ETIME=100 XSIZE=100 YSIZE=100 CBIN=1     RBIN=1   CORG=0    RORG=0    DARK=0 LO
      OP=1
439 //ETIME=100 XSIZE=2184 YSIZE=1472 CBIN=1 RBIN=1 CORG=0 RORG=0 DARK=0
440         rx=sscanf ( mensaje,"ETIME=%d XSIZE=%d YSIZE=%d CBIN=%d RBIN=%d  CORG=%d
      RORG=%d DARK=%d "\
441                     ,&ccd.etime,&ccd.cols,&ccd.rows,&ccd.cbin,&ccd.rbin,&ccd.
      corg,&ccd.rorg,&ccd.dark );
442         cout <<"Tengo "<<rx<<" Parametros \n";
443 
444         ccd.showinfo();
445 
446         if ( !ccd.ccd_init )
447         {
448                 cout <<"No has hecho el INIT CCDTYPE ....SOB \n";
449                 ccd.init ( myfd );
450         }
451         /*
452                 // revisa que el numero de columnas y renglones sea par
453                 if ( ccd.cols % 2 != 0 ) ccd.cols--; //quitarle un numero para ha
      cerlo par
454                 if ( ccd.rows % 2 != 0 ) ccd.rows--; //quitarle un numero para ha
      cerlo par
455         */
456         ccd.update();   //actualiza varios parametros
457 
458         if ( ccd.maxpixels==ccd.imagePixels )
```

```
459         {
460                 ccd.full_image=TRUE;
461                 cout << "Full image Request!" <<endl;
462         }
463         else
464         {
465                 ccd.full_image=FALSE;
466                 cout << "ROI image Request!" <<endl;
467         }
468         cout<<"Max pixels= "<<ccd.maxpixels<<endl;
469         cout<<"Img pixels= "<<ccd.imagePixels<<endl;
470
471 //*************************************
472         ccd.cancela=!ccd.expose ( ccd.etime,myfd );
473
474         if ( !ccd.cancela )
475         {
476                 sprintf ( mensaje,"BINARIO %d \n",ccd.imagePixels );
477                 red.responde_cliente ( mensaje,myfd );
478
479                 //mandar la temperatura
480                 t=ccd.lee_temperatura();
481                 sprintf ( mensaje,"LEE_TEMP %2.1f\n",t );
482                 red.responde_cliente ( mensaje,myfd );
483
484                 //V0.97
485                 if ( ccd.do_deinterlace )
486                 {
487                         cout <<"Vamos hacer deinterlace por la lectura DUAL"<<end
    l;
488                         error=ccd.deinterlace ( ccd.cols, ccd.rows );
489
490                         if ( error<0 )
491                         {
492                                 cout <<"deinterlace error= "<<ccd.error_string<<e
    ndl;
493                         }
494                 } else cout <<"Usamos solo un canal de salida \n";
495
496                 if ( ccd.do_estadistica )
497                 {
498                         ccd.estadistica ( &minn,&maxx,&mean, &stdev, &saturated )
    ;
499                         //printf ( "Estadistica min=%d, max=%d, mean=%3.2f stdev=
    %3.2f saturated= %d\n",minn,maxx,mean,stdev,saturated );
500                         sprintf ( mensaje,"ESTADISTICA min=%d, max=%d, mean=%3.2f
     stdev=%3.2f saturated=%d\n",minn,maxx,mean,stdev,saturated );
501                         //cout <<mensaje;
502                         red.responde_cliente ( mensaje,myfd );
503                 }  else cout <<"No vamos hacer estadistica \n";
504         }
505         else
506         {
507                 strcpy ( mensaje,"CANCELADO \n" );
508                 cout <<mensaje;
509                 red.responde_cliente ( mensaje,myfd );
510         }
511
512 //*************************************
513         ccd.cancela=false;
514         printf ( "\n Expose DONE! \n" );
515         red.responde_cliente ( ( char * )  "TERMINE \n",myfd );
516 }
```

Here is the call graph for this function:

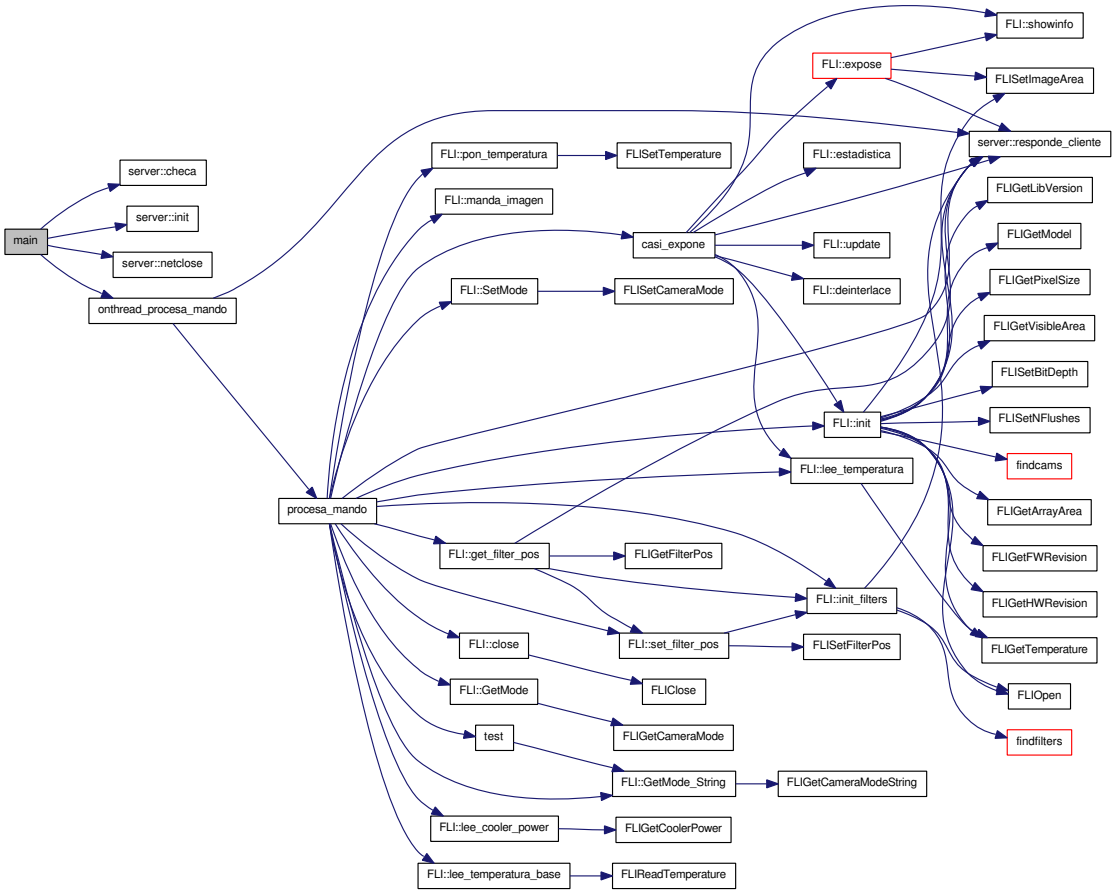


4.5.2.2 int main (int *argc*, char * *argv*[ ])

```
118 {
119         //cam_t *cam = NULL;
120         int mycliente;
121         pthread_attr_t tattr;
122 
123         system ( "clear" );
124         cout << "\nUNAM-IA-OAN ENSENADA" << endl;
```

```
125         cout << "\nHello, world!, FLI CaM Server is running, "<<MyVersion  << end
      l<< endl;
126         cout << "2011 By E. Colorado  " << endl;
127
128         red.procesando=FALSE;   //control de procesos en thread
129         red.busy=FALSE;
130
131         //inializar RED tipo servidor
132         red.salida=false;
133         red.init ( PORT );
134         cout <<"Ready using Sever port: "<<PORT<<endl;
135         cout <<"looping...";
136         red.op=111; //modificar para que no se cicle
137
138
139         //thread y mutex
140         res=pthread_mutex_init ( &mutex_busy,NULL );
141         if ( res!=0 )
142         {
143                 perror ( "Fallo Mutex" );
144                 exit ( EXIT_FAILURE );
145         }
146
147         //atributos del thread
148         pthread_attr_init ( &tattr );
149         pthread_attr_setdetachstate ( &tattr,PTHREAD_CREATE_DETACHED );
150
151         do
152         {
153                 cout << "main red checa \n";
154                 mycliente=red.checa();  //espera aqui hast que exista dato
155                 //cout << "Recibi de red= "<<red.instruccion<<endl;
156
157                 //generar nuevo thread
158                 res=pthread_create ( &thread_mando,&tattr,onthread_procesa_mando,
       ( void * ) mycliente );
159                 if ( res!=0 )
160                 {
161                         perror ( "Fallo Thread onthread_procesa_mando" );
162                         exit ( EXIT_FAILURE );
163                 }
164                 cout <<"salida= "<<red.salida<<endl;
165                 cout <<"Thread onthread_procesa_mando OK"<<endl;
166
167         }
168         while ( !red.salida );
169
170 //cerrar todo
171         pthread_mutex_destroy ( &mutex_busy );
172         cout << "mutex destroy \n" ;
173         red.netclose();
174         cout << "red close \n" ;
175
176         cout <<"\n FLI Server DONE! \n";
177
178         return EXIT_SUCCESS;
179 }
```

Here is the call graph for this function:

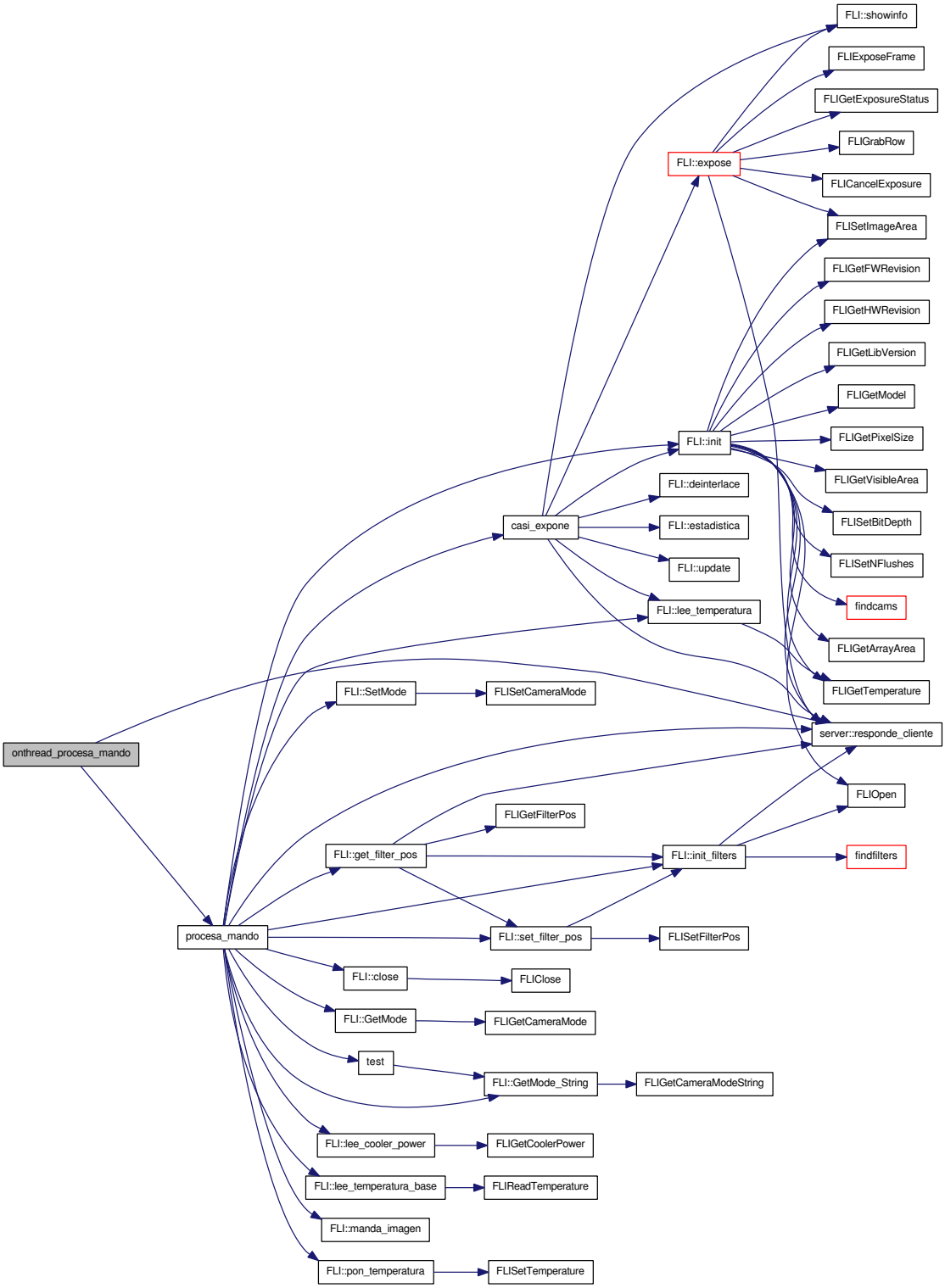


4.5.2.3 void * onthread_procesa_mando (void * *arg*)

```
74 {
75         int myfd;
76         char txt[200];
77         int local;
78         int i;
79         bool resp=TRUE;
80 
81         //ignorar senales
82         signal ( SIGCHLD, SIG_IGN );
83         signal ( SIGPIPE, SIG_IGN );
84 
85         //traernos cliente nuevo de los argumentos
86         local= ( int ) arg;
87         myfd=local;
88         cout <<"Cliente local es "<<myfd <<endl;
89 
90         memset ( txt,0,200 );
91 
92         //if ( ccd.debug ) cout <<"En thread procesa_mando"<<endl;
93         //LEER LO DE LA RED
94         cout <<"Voy a leer de red hasta "<<sizeof ( txt ) <<" char "<<endl;
95         i=read ( myfd,&txt,sizeof ( txt ) );
96         cout <<"Lei de red "<<i<< " ->"<<txt<<endl;
```

```
97
98          resp=procesa_mando ( myfd,txt );
99
100          //cierra conexion del cliente
101          if ( resp )
102                  red.responde_cliente ( ( char * )  "CLOSE \n",myfd );
103
104          usleep ( 200000 );
105          cout <<"Cerrando socket con cliente...\n";
106          close ( myfd );
107
108          cout << "--------------------------------"<<endl ;
109
110          //if ( ccd.debug ) cout << "thread procesa_mando exit \n" ;
111          //memset ( txt,0,200 );
112          //esperar un rato a que cierre el socket
113          sleep ( 5 );
114          pthread_exit ( 0 );
115 }
```

Here is the call graph for this function:



4.5.2.4 int procesa_mando (int *myfd*, char * *txt*)

```
183 {
184         static char *comando,*parametro;
185         int r,j,i;
186         char mensaje[200];
187         double f;
188         int resp =TRUE;
189 
190         cout <<myfd<<" <- "<<txt<<endl;
191         r=strlen ( txt );
192         cout <<"llegaron "<<r<<" datos \n";
193         if ( r<3 )
194         {
195                 cout <<"No llegaron suficientes datos !!!!!!!\n";
196                 return resp;
197         }
198 
199         //pasar los datos a mayusculas
200         for ( i = 0; txt[i]; i++ )
201                 txt[i] = toupper ( txt[i] );
202 
203         parametro=NULL; //para que no haya basura
204         comando=NULL;
205         comando = strtok ( txt," " );
206         cout <<"comando="<<comando<<endl;
207         parametro=strtok ( NULL,"\n" );
208         if ( parametro!=NULL )
209         {
210                 cout <<"parametros="<<parametro<<endl;
211         }
212 
213         r=strlen ( comando );
214         cout <<r<<" lenght"<<endl;
215 
216 
218         red.op=100;
219         j=strcmp ( comando,"SALIR" );if ( !j )          red.op=0;
220         j=strcmp ( comando,"INIT" );    if ( !j )       red.op=1;
221         j=strcmp ( comando,"EXPONE" );  if ( !j )       red.op=2;
222         j=strcmp ( comando,"GET_TEMP" );if ( !j )       red.op=3;
223         j=strcmp ( comando,"LEE_TEMP" );if ( !j )       red.op=3;
224         j=strcmp ( comando,"SET_TEMP" );if ( !j )       red.op=4;
225         //j=strcmp ( comando,"GET_FULL" );if ( !j )     red.op=5;
226         j=strcmp ( comando,"MANDABIN" );if ( !j )       red.op=6;
227         j=strcmp ( comando,"CLOSE" );   if ( !j )       red.op=7;
228         j=strcmp ( comando,"CANCELA" ); if ( !j )       red.op=8;
229         j=strcmp ( comando,"STATUS" ); if ( !j )        red.op=9;
230         j=strcmp ( comando,"ESTATUS" ); if ( !j )       red.op=9;
231         j=strcmp ( comando,"TEST" );    if ( !j )       red.op=10;
232         //modos de salida y velocidad de la camara
233         j=strcmp ( comando,"SET_CAM_MODE" );    if ( !j ) red.op=11;
234         j=strcmp ( comando,"GET_CAM_MODE" );    if ( !j ) red.op=12;
235         j=strcmp ( comando,"GET_CAM_STRING" );  if ( !j ) red.op=13;
236         //V2.1
237         j=strcmp ( comando,"LEE_TEMP_BASE" );if ( !j )  red.op=14;
238         j=strcmp ( comando,"LEE_POWER" );if ( !j )      red.op=15;
239         //V2.12
240         j=strcmp ( comando,"INIT_FILTERS" );if ( !j )   red.op=16;
241         j=strcmp ( comando,"SET_FILTER_POS" );if ( !j ) red.op=17;
242         j=strcmp ( comando,"GET_FILTER_POS" );if ( !j ) red.op=18;
243 
244 
245         printf ( " ...(red.op=%d)\n",red.op );
246         switch ( red.op )
247         {
248                 case 0:
```

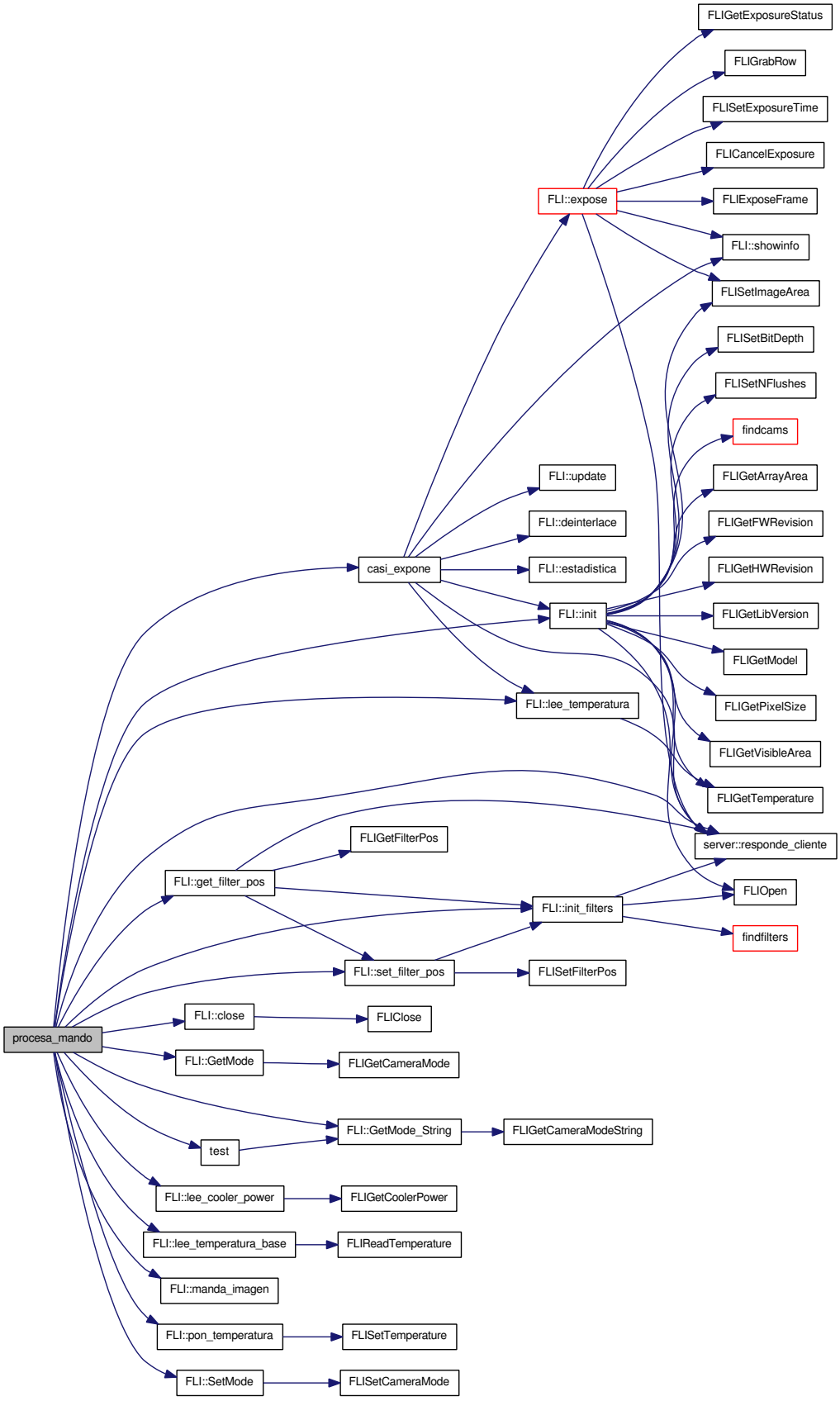
```
249                 {
250                         printf ( "Hay que salir \n" );
251                         red.salida=true;
252                         ccd.close();
253                         break;
254                 }
255                 case 1:
256                 {
257                         printf ( "INIT \n" );
258                         ccd.init ( myfd );
259                         break;
260                 }
261
262                 case 2:
263                 {
264                         //EXPONE
265                         //realmente no necesito el mutex ?
266                         printf ( "expone \n" );
267                         pthread_mutex_lock ( &mutex_busy );
268                         ccd.busy=TRUE;
269                         pthread_mutex_unlock ( &mutex_busy );
270
271                         casi_expone ( parametro,myfd );
272
273                         pthread_mutex_lock ( &mutex_busy );
274                         ccd.busy=FALSE;
275                         pthread_mutex_unlock ( &mutex_busy );
276
277                         break;
278                 }
279                 case 3:
280                 {       //temp
281                         cout << "Leyendo temperatura del CCD" << endl;
282                         if ( !ccd.busy )
283                         {
284                                 f = ccd.lee_temperatura();
285                         }
286                         else
287                         {
288                                 f=ccd.ccdtemp;
289                                 cout <<"CCD ocupado, te voy a dar la Temp. anteri
    or.. \n";
290                         }
291                         sprintf ( mensaje,"LEE_TEMP %2.1f\n",f );
292                         red.responde_cliente ( mensaje,myfd );
293                         cout << "CCD Temp=" << f<<endl;
294
295                         break;
296                 }
297                 case 4:
298                 {
299                         printf ( "set temp \n" );
300                         sscanf ( parametro,"%lf",&f );
301                         printf ( "set temp to %3.2lf\n",f );
302                         ccd.pon_temperatura ( f );
303                         break;
304                 }
305                 case 6:
306                 {
307                         //MANDABIN
308                         cout <<"Vamos hacer mandabin \n";
309                         ccd.manda_imagen ( myfd );
310                         resp=FALSE;
311                         break;
312                 }
313                 case 7:
314                 {
```

```
315                         printf ( "close fli driver \n" );
316                         ccd.close( );
317                         break;
318                 }
319                 case 8:
320                 {
321                         cout<<"CANCELANDO!!!!!!!!!!!!!!!!!!"<<endl;
322                         ccd.cancela=TRUE;
323                         break;
324                 }
325                 case 9:
326                 {
327                         cout<<"Mandando Status:"<<endl;
328                         if ( ccd.busy ) strcpy ( mensaje,"BUSY \n" ); else strcpy
     ( mensaje,"READY \n" );
329                         red.responde_cliente ( mensaje,myfd );
330                         break;
331                 }
332                 case 10:
333                         test();
334                         break;
335                 case 11:
336                         printf ( "set cam mode  \n" );
337                         sscanf ( parametro,"%d",&i );
338                         printf ( "set cam mode  %d\n",i );
339                         ccd.SetMode ( i );
340                 case 12:
341                         printf ( "get cam mode  \n" );
342                         i=ccd.GetMode();
343                         sprintf ( mensaje,"CAM_MODE %d\n",i );
344                         cout <<mensaje;
345                         red.responde_cliente ( mensaje,myfd );
346                         break;
347                 case 13:
348                         sscanf ( parametro,"%d",&i );
349                         printf ( "get cam mode  string %d\n",i );
350
351                         char *t;
352                         t=ccd.GetMode_String ( i );
353
354                         sprintf ( mensaje,"CAM_MODE_STRING %s\n",t );
355                         cout <<mensaje;
356                         red.responde_cliente ( mensaje,myfd );
357                         break;
358                 case 14:
359                 {       //temp_base
360                         cout << "Leyendo temperatura de la base del CCD" << endl;

361                         if ( !ccd.busy )
362                         {
363                                 f = ccd.lee_temperatura_base();
364                         }
365                         else
366                         {
367                                 f=ccd.basetemp;
368                                 cout <<"CCD ocupado, te voy a dar la Temp. base a
     nterior.. \n";
369                         }
370                         sprintf ( mensaje,"LEE_TEMP_BASE %2.1f\n",f );
371                         red.responde_cliente ( mensaje,myfd );
372                         cout << "CCD Base Temp=" << f<<endl;
373
374                         break;
375                 }
376                 case 15:
377                 {       //power
378                         cout << "Leyendo el cooler power" << endl;
```

```
379                         if ( !ccd.busy )
380                         {
381                                 f = ccd.lee_cooler_power();
382                         }
383                         else
384                         {
385                                 f=ccd.power;
386                                 cout <<"CCD ocupado, te voy a dar el power anteri
    or.. \n";
387                         }
388                         sprintf ( mensaje,"LEE_POWER %2.1f\n",f );
389                         red.responde_cliente ( mensaje,myfd );
390                         cout << "CCD cooler power=" << f<<endl;
391
392                         break;
393                 }
394                 case 16:
395                         printf ( "INIT Filters\n" );
396                         ccd.init_filters ( myfd );
397                         break;
398                 case 17:
399                         printf ( "Move  Filters ... \n" );
400                         sscanf ( parametro,"%d",&i );
401                         printf ( "to: %d \n",i );
402                         ccd.set_filter_pos(myfd,i);
403                         break;
404                 case 18:
405                         printf ( "get  Filters pos ... \n" );
406                         ccd.get_filter_pos(myfd);
407                         break;
408                 case 100:
409                 {
410
411                         printf ( "!!!!!Mando no Reconocido... \n" );
412                         strcpy ( mensaje,"!!!!!Mando no Reconocido... \n" );
413                         red.responde_cliente ( mensaje,myfd );
414                         break;
415                 }
416
417         }
418         ; //end switch
419
420         return resp;
421
422 }
```

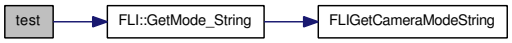
Here is the call graph for this function:



4.5.2.5 void test ()

```
519 {
520
521         cout <<"modo 0 "<<ccd.GetMode_String ( 0 ) <<endl;
522         cout <<"modo 1 "<<ccd.GetMode_String ( 1 ) <<endl;
523         cout <<"modo 2 "<<ccd.GetMode_String ( 2 ) <<endl;
524         //cout <<"modo 3 "<<ccd.GetMode_String(3)<<endl;
525
526
527 }
```

Here is the call graph for this function:



4.5.3 Variable Documentation

4.5.3.1 FLI ccd

4.5.3.2 list<micliente>::iterator i

4.5.3.3 list<micliente> lista

4.5.3.4 pthread_mutex_t mutex_busy

4.5.3.5 server red

4.5.3.6 int res

4.5.3.7 void* t_result

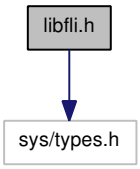
4.5.3.8 pthread_t thread_mando

4.5.3.9 pthread_t thread_red

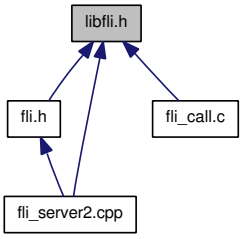
4.6 libfli.h File Reference

```
#include <sys/types.h>
```

Include dependency graph for libfli.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define DllImport __declspec(dllimport)
- #define DllExport __declspec(dllexport)
- #define FLI_INVALID_DEVICE (-1)
- #define FLIDOMAIN_NONE (0x00)
- #define FLIDOMAIN_PARALLEL_PORT (0x01)
- #define FLIDOMAIN_USB (0x02)
- #define FLIDOMAIN_SERIAL (0x03)
- #define FLIDOMAIN_INET (0x04)
- #define FLIDOMAIN_SERIAL_19200 (0x05)
- #define FLIDOMAIN_SERIAL_1200 (0x06)
- #define FLIDEVICE_NONE (0x000)
- #define FLIDEVICE_CAMERA (0x100)
- #define FLIDEVICE_FILTERWHEEL (0x200)
- #define FLIDEVICE_FOCUSER (0x300)
- #define FLIDEVICE_HS_FILTERWHEEL (0x0400)
- #define FLIDEVICE_RAW (0x0f00)
- #define FLIDEVICE_ENUMERATE_BY_CONNECTION (0x8000)
- #define FLI_FRAME_TYPE_NORMAL (0)
- #define FLI_FRAME_TYPE_DARK (1)
- #define FLI_FRAME_TYPE_FLOOD (2)
- #define FLI_FRAME_TYPE_RBI_FLUSH (FLI_FRAME_TYPE_FLOOD | FLI_FRAME_TYPE_DARK)
- #define FLI_MODE_8BIT (0)
- #define FLI_MODE_16BIT (1)

- #define FLI_SHUTTER_CLOSE (0x0000)
- #define FLI_SHUTTER_OPEN (0x0001)
- #define FLI_SHUTTER_EXTERNAL_TRIGGER (0x0002)
- #define FLI_SHUTTER_EXTERNAL_TRIGGER_LOW (0x0002)
- #define FLI_SHUTTER_EXTERNAL_TRIGGER_HIGH (0x0004)
- #define FLI_SHUTTER_EXTERNAL_EXPOSURE_CONTROL (0x0008)
- #define FLI_BGFLUSH_STOP (0x0000)
- #define FLI_BGFLUSH_START (0x0001)
- #define FLI_TEMPERATURE_INTERNAL (0x0000)
- #define FLI_TEMPERATURE_EXTERNAL (0x0001)
- #define FLI_TEMPERATURE_CCD (0x0000)
- #define FLI_TEMPERATURE_BASE (0x0001)
- #define FLI_CAMERA_STATUS_UNKNOWN (0xffffffff)
- #define FLI_CAMERA_STATUS_MASK (0x000000ff)
- #define FLI_CAMERA_STATUS_IDLE (0x00)
- #define FLI_CAMERA_STATUS_WAITING_FOR_TRIGGER (0x01)
- #define FLI_CAMERA_STATUS_EXPOSING (0x02)
- #define FLI_CAMERA_STATUS_READING_CCD (0x03)
- #define FLI_CAMERA_DATA_READY (0x80000000)
- #define FLI_FOCUSER_STATUS_UNKNOWN (0xffffffff)
- #define FLI_FOCUSER_STATUS_HOMING (0x00000004)
- #define FLI_FOCUSER_STATUS_MOVING_IN (0x00000001)
- #define FLI_FOCUSER_STATUS_MOVING_OUT (0x00000002)
- #define FLI_FOCUSER_STATUS_MOVING_MASK (0x00000007)
- #define FLI_FOCUSER_STATUS_HOME (0x00000080)
- #define FLI_FOCUSER_STATUS_LIMIT (0x00000040)
- #define FLI_FOCUSER_STATUS_LEGACY (0x10000000)
- #define FLIDEBUG_NONE (0x00)
- #define FLIDEBUG_INFO (0x01)
- #define FLIDEBUG_WARN (0x02)
- #define FLIDEBUG_FAIL (0x04)
- #define FLIDEBUG_ALL (FLIDEBUG_INFO | FLIDEBUG_WARN | FLIDEBUG_FAIL)
- #define FLI_IO_P0 (0x01)
- #define FLI_IO_P1 (0x02)
- #define FLI_IO_P2 (0x04)
- #define FLI_IO_P3 (0x08)
- #define FLI_FAN_SPEED_OFF (0x00)
- #define FLI_FAN_SPEED_ON (0xffffffff)
- #define LIBFLIAPI long

Typedefs

- typedef long flidev_t
- typedef long flidomain_t
- typedef long fliframe_t
- typedef long flibitdepth_t
- typedef long flishutter_t
- typedef long flibgflush_t
- typedef long flichannel_t
- typedef long flidebug_t

- typedef long flimode_t
- typedef long flistatus_t
- typedef long flitdirate_t
- typedef long flitdiflags_t

Functions

- LIBFLIAPI FLIOpen (flidev_t *dev, char *name, flidomain_t domain)
- LIBFLIAPI FLISetDebugLevel (char *host, flidebug_t level)
- LIBFLIAPI FLIClose (flidev_t dev)
- LIBFLIAPI FLIGetLibVersion (char *ver, size_t len)
- LIBFLIAPI FLIGetModel (flidev_t dev, char *model, size_t len)
- LIBFLIAPI FLIGetPixelSize (flidev_t dev, double *pixel_x, double *pixel_y)
- LIBFLIAPI FLIGetHWRevision (flidev_t dev, long *hwrev)
- LIBFLIAPI FLIGetFWRevision (flidev_t dev, long *fwrev)
- LIBFLIAPI FLIGetArrayArea (flidev_t dev, long *ul_x, long *ul_y, long *lr_x, long *lr_y)
- LIBFLIAPI FLIGetVisibleArea (flidev_t dev, long *ul_x, long *ul_y, long *lr_x, long *lr_y)
- LIBFLIAPI FLISetExposureTime (flidev_t dev, long exptime)
- LIBFLIAPI FLISetImageArea (flidev_t dev, long ul_x, long ul_y, long lr_x, long lr_y)
- LIBFLIAPI FLISetHBin (flidev_t dev, long hbin)
- LIBFLIAPI FLISetVBin (flidev_t dev, long vbin)
- LIBFLIAPI FLISetFrameType (flidev_t dev, fliframe_t frametype)
- LIBFLIAPI FLICancelExposure (flidev_t dev)
- LIBFLIAPI FLIGetExposureStatus (flidev_t dev, long *timeleft)
- LIBFLIAPI FLISetTemperature (flidev_t dev, double temperature)
- LIBFLIAPI FLIGetTemperature (flidev_t dev, double *temperature)
- LIBFLIAPI FLIGetCoolerPower (flidev_t dev, double *power)
- LIBFLIAPI FLIGrabRow (flidev_t dev, void *buff, size_t width)
- LIBFLIAPI FLIExposeFrame (flidev_t dev)
- LIBFLIAPI FLIFlushRow (flidev_t dev, long rows, long repeat)
- LIBFLIAPI FLISetNFlushes (flidev_t dev, long nflushes)
- LIBFLIAPI FLISetBitDepth (flidev_t dev, flibitdepth_t bitdepth)
- LIBFLIAPI FLIReadIOPort (flidev_t dev, long *ioportset)
- LIBFLIAPI FLIWriteIOPort (flidev_t dev, long ioportset)
- LIBFLIAPI FLIConfigureIOPort (flidev_t dev, long ioportset)
- LIBFLIAPI FLILockDevice (flidev_t dev)
- LIBFLIAPI FLIUnlockDevice (flidev_t dev)
- LIBFLIAPI FLIControlShutter (flidev_t dev, flishutter_t shutter)
- LIBFLIAPI FLIControlBackgroundFlush (flidev_t dev, flibgflush_t bgflush)
- LIBFLIAPI FLISetDAC (flidev_t dev, unsigned long dacset)
- LIBFLIAPI FLIList (flidomain_t domain, char ***names)
- LIBFLIAPI FLIFreeList (char **names)
- LIBFLIAPI FLISetFilterPos (flidev_t dev, long filter)
- LIBFLIAPI FLIGetFilterPos (flidev_t dev, long *filter)
- LIBFLIAPI FLIGetFilterCount (flidev_t dev, long *filter)
- LIBFLIAPI FLIStepMotor (flidev_t dev, long steps)
- LIBFLIAPI FLIStepMotorAsync (flidev_t dev, long steps)
- LIBFLIAPI FLIGetStepperPosition (flidev_t dev, long *position)
- LIBFLIAPI FLIGetStepsRemaining (flidev_t dev, long *steps)
- LIBFLIAPI FLIHomeFocuser (flidev_t dev)

- LIBFLIAPI FLICreateList (flidomain_t domain)
- LIBFLIAPI FLIDeleteList (void)
- LIBFLIAPI FLIListFirst (flidomain_t *domain, char *filename, size_t fnlen, char *name, size_t namelen)
- LIBFLIAPI FLIListNext (flidomain_t *domain, char *filename, size_t fnlen, char *name, size_t namelen)
- LIBFLIAPI FLIReadTemperature (flidev_t dev, flichannel_t channel, double *temperature)
- LIBFLIAPI FLIGetFocuserExtent (flidev_t dev, long *extent)
- LIBFLIAPI FLIUsbBulkIO (flidev_t dev, int ep, void *buf, long *len)
- LIBFLIAPI FLIGetDeviceStatus (flidev_t dev, long *status)
- LIBFLIAPI FLIGetCameraModeString (flidev_t dev, flimode_t mode_index, char *mode_string, size_t siz)
- LIBFLIAPI FLIGetCameraMode (flidev_t dev, flimode_t *mode_index)
- LIBFLIAPI FLISetCameraMode (flidev_t dev, flimode_t mode_index)
- LIBFLIAPI FLIHomeDevice (flidev_t dev)
- LIBFLIAPI FLIGrabFrame (flidev_t dev, void *buff, size_t buffsize, size_t *bytesgrabbed)
- LIBFLIAPI FLISetTDI (flidev_t dev, flitdirate_t tdi_rate, flitdiflags_t flags)
- LIBFLIAPI FLIGrabVideoFrame (flidev_t dev, void *buff, size_t size)
- LIBFLIAPI FLIStopVideoMode (flidev_t dev)
- LIBFLIAPI FLIStartVideoMode (flidev_t dev)
- LIBFLIAPI FLIGetSerialString (flidev_t dev, char *serial, size_t len)
- LIBFLIAPI FLIEndExposure (flidev_t dev)
- LIBFLIAPI FLITriggerExposure (flidev_t dev)
- LIBFLIAPI FLISetFanSpeed (flidev_t dev, long fan_speed)

4.6.1 Define Documentation

4.6.1.1 #define DllExport __declspec(dllexport)

4.6.1.2 #define DllImport __declspec(dllimport)

4.6.1.3 #define FLI_BGFLUSH_START (0x0001)

4.6.1.4 #define FLI_BGFLUSH_STOP (0x0000)

4.6.1.5 #define FLI_CAMERA_DATA_READY (0x80000000)

4.6.1.6 #define FLI_CAMERA_STATUS_EXPOSING (0x02)

4.6.1.7 #define FLI_CAMERA_STATUS_IDLE (0x00)

4.6.1.8 #define FLI_CAMERA_STATUS_MASK (0x000000ff)

4.6.1.9 #define FLI_CAMERA_STATUS_READING_CCD (0x03)

4.6.1.10 #define FLI_CAMERA_STATUS_UNKNOWN (0xffffffff)

4.6.1.11 #define FLI_CAMERA_STATUS_WAITING_FOR_TRIGGER (0x01)

4.6.1.12 #define FLI_FAN_SPEED_OFF (0x00)

4.6.1.13 #define FLI_FAN_SPEED_ON (0xffffffff)

4.6.1.14 #define FLI_FOCUSER_STATUS_HOME (0x00000080)

4.6.1.15 #define FLI_FOCUSER_STATUS_HOMING (0x00000004)

4.6.1.16 #define FLI_FOCUSER_STATUS_LEGACY (0x10000000)

4.6.1.17 #define FLI_FOCUSER_STATUS_LIMIT (0x00000040)

4.6.1.18 #define FLI_FOCUSER_STATUS_MOVING_IN (0x00000001)

4.6.1.19 #define FLI_FOCUSER_STATUS_MOVING_MASK (0x00000007)

4.6.1.20 #define FLI_FOCUSER_STATUS_MOVING_OUT (0x00000002)

4.6.1.21 #define FLI_FOCUSER_STATUS_UNKNOWN (0xffffffff)

4.6.1.22 #define FLI_FRAME_TYPE_DARK (1)

4.6.1.23 #define FLI_FRAME_TYPE_FLOOD (2)

4.6.1.24 #define FLI_FRAME_TYPE_NORMAL (0)

4.6.1.25 #define FLI_FRAME_TYPE_RBI_FLUSH (FLI_FRAME_TYPE_FLOOD | FLI_FRAME_TYPE_DARK)

4.6.1.26 #define FLI_INVALID_DEVICE (-1)

An opaque handle used by library functions to refer to FLI hardware.

4.6.1.27 #define FLI_IO_P0 (0x01)

4.6.1.28 #define FLI_IO_P1 (0x02)

4.6.1.29 #define FLI_IO_P2 (0x04)

4.6.1.30 #define FLI_IO_P3 (0x08)

4.6.1.31 #define FLI_MODE_16BIT (1)

4.6.1.32 #define FLI_MODE_8BIT (0)

4.6.1.33 #define FLI_SHUTTER_CLOSE (0x0000)

4.6.1.34 #define FLI_SHUTTER_EXTERNAL_EXPOSURE_CONTROL (0x0008)

4.6.1.35 #define FLI_SHUTTER_EXTERNAL_TRIGGER (0x0002)

4.6.1.36 #define FLI_SHUTTER_EXTERNAL_TRIGGER_HIGH (0x0004)

4.6.1.37 #define FLI_SHUTTER_EXTERNAL_TRIGGER_LOW (0x0002)

4.6.1.38 #define FLI_SHUTTER_OPEN (0x0001)

4.6.1.39 #define FLI_TEMPERATURE_BASE (0x0001)

4.6.1.40 #define FLI_TEMPERATURE_CCD (0x0000)

4.6.1.41 #define FLI_TEMPERATURE_EXTERNAL (0x0001)

4.6.1.42 #define FLI_TEMPERATURE_INTERNAL (0x0000)

4.6.1.43 #define FLIDEBUG_ALL (FLIDEBUG_INFO | FLIDEBUG_WARN | FLIDEBUG_FAIL)

4.6.1.44 #define FLIDEBUG_FAIL (0x04)

4.6.1.45 #define FLIDEBUG_INFO (0x01)

4.6.1.46 #define FLIDEBUG_NONE (0x00)

4.6.1.47 #define FLIDEBUG_WARN (0x02)

4.6.1.48 #define FLIDEVICE_CAMERA (0x100)

4.6.1.49 #define FLIDEVICE_ENUMERATE_BY_CONNECTION (0x8000)

4.6.1.50 #define FLIDEVICE_FILTERWHEEL (0x200)

4.6.1.51 #define FLIDEVICE_FOCUSER (0x300)

4.6.1.52 #define FLIDEVICE_HS_FILTERWHEEL (0x0400)

4.6.1.53 #define FLIDEVICE_NONE (0x000)

4.6.1.54 #define FLIDEVICE_RAW (0x0f00)

4.6.1.55 #define FLIDOMAIN_INET (0x04)

4.6.1.56 #define FLIDOMAIN_NONE (0x00)

See also

FLIControlBackgroundFlush

4.6.2.2 typedef long flibitdepth_t

The gray-scale bit depth for an FLI camera device. Valid bit depths are {FLI_MODE_8BIT} and {FLI_MODE_16BIT}.

See also

FLISetBitDepth

4.6.2.3 typedef long flichannel_t

Type used to determine which temperature channel to read. Valid channel types are {FLI_TEMPERATURE_INTERNAL} and {FLI_TEMPERATURE_EXTERNAL}.

See also

FLIReadTemperature

4.6.2.4 typedef long flidebug_t

Type specifying library debug levels. Valid debug levels are {FLIDEBUG_NONE}, {FLIDEBUG_INFO}, {FLIDEBUG_WARN}, and {FLIDEBUG_FAIL}.

See also

FLISetDebugLevel

4.6.2.5 typedef long flidev_t

4.6.2.6 typedef long flidomain_t

The domain of an FLI device. This consists of a bitwise ORed combination of interface method and device type. Valid interfaces are {FLIDOMAIN_PARALLEL_PORT}, {FLIDOMAIN_USB}, {FLIDOMAIN_SERIAL}, and {FLIDOMAIN_INET}. Valid device types are {FLIDEVICE_CAMERA}, {FLIDOMAIN_FILTERWHEEL}, and {FLIDOMAIN_FOCUSER}.

See also

FLIOpen
FLIList

4.6.2.7 typedef long fliframe_t

The frame type for an FLI CCD camera device. Valid frame types are {FLI_FRAME_TYPE_NORMAL} and {FLI_FRAME_TYPE_DARK}.

See also

FLISetFrameType

4.6.2.8 typedef long flimode_t

4.6.2.9 typedef long flishutter_t

Type used for shutter operations for an FLI camera device. Valid shutter types are {FLI_SHUTTER_CLOSE}, {FLI_SHUTTER_OPEN}, {FLI_SHUTTER_EXTERNAL_TRIGGER}, {FLI_SHUTTER_EXTERNAL_TRIGGER_LOW}, and {FLI_SHUTTER_EXTERNAL_TRIGGER_HIGH}.

See also

FLIControlShutter

4.6.2.10 typedef long flistatus_t

4.6.2.11 typedef long flitdiflags_t

4.6.2.12 typedef long flitdirate_t

4.6.3 Function Documentation

4.6.3.1 LIBFLIAPI FLICancelExposure (flidev_t *dev*)

4.6.3.2 LIBFLIAPI FLIClose (flidev_t *dev*)

4.6.3.3 LIBFLIAPI FLIConfigureIOPort (flidev_t *dev*, long *ioportset*)

4.6.3.4 LIBFLIAPI FLIControlBackgroundFlush (flidev_t *dev*, flibgflush_t *bgflush*)

4.6.3.5 LIBFLIAPI FLIControlShutter (flidev_t *dev*, flishutter_t *shutter*)

4.6.3.6 LIBFLIAPI FLICreateList (flidomain_t *domain*)

4.6.3.7 LIBFLIAPI FLIDeleteList (void)

4.6.3.8 LIBFLIAPI FLIEndExposure (flidev_t *dev*)

4.6.3.9 LIBFLIAPI FLIExposeFrame (flidev_t *dev*)

4.6.3.10 LIBFLIAPI FLIFlushRow (flidev_t *dev*, long *rows*, long *repeat*)

4.6.3.11 LIBFLIAPI FLIFreeList (char ** *names*)

4.6.3.12 LIBFLIAPI FLIGetArrayArea (flidev_t *dev*, long * *ul_x*, long * *ul_y*, long * *lr_x*, long * *lr_y*)

4.6.3.13 LIBFLIAPI FLIGetCameraMode (flidev_t *dev*, flimode_t * *mode_index*)

4.6.3.14 LIBFLIAPI FLIGetCameraModeString (flidev_t *dev*, flimode_t *mode_index*, char * *mode_string*, size_t *siz*)

4.6.3.15 LIBFLIAPI FLIGetCoolerPower (flidev_t *dev*, double * *power*)

4.6.3.16 LIBFLIAPI FLIGetDeviceStatus (flidev_t *dev*, long * *status*)

4.6.3.17 LIBFLIAPI FLIGetExposureStatus (flidev_t *dev*, long * *timeleft*)

4.6.3.18 LIBFLIAPI FLIGetFilterCount (flidev_t *dev*, long * *filter*)

4.6.3.19 LIBFLIAPI FLIGetFilterPos (flidev_t *dev*, long * *filter*)

4.6.3.20 LIBFLIAPI FLIGetFocuserExtent (flidev_t *dev*, long * *extent*)

4.6.3.21 LIBFLIAPI FLIGetFWRevision (flidev_t *dev*, long * *fwrev*)

4.6.3.22 LIBFLIAPI FLIGetHWRevision (flidev_t *dev*, long * *hwrev*)

4.6.3.23 LIBFLIAPI FLIGetLibVersion (char * *ver*, size_t *len*)

4.6.3.24 LIBFLIAPI FLIGetModel (flidev_t *dev*, char * *model*, size_t *len*)

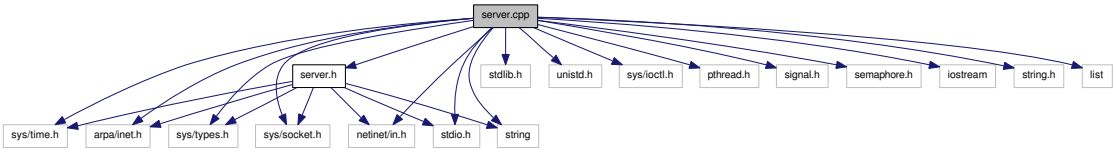
4.6.3.25 LIBFLIAPI FLIGetPixelSize (flidev_t *dev*, double * *pixel_x*, double * *pixel_y*)

4.6.3.26 LIBFLIAPI FLIGetSerialString (flidev_t *dev*, char * *serial*, size_t *len*)

4.7 server.cpp File Reference

```
#include "server.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <semaphore.h>
#include <iostream>
#include <string.h>
#include <string>
#include <list>
```

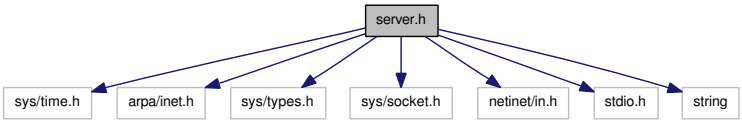
Include dependency graph for server.cpp:



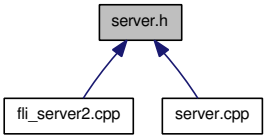
4.8 server.h File Reference

```
#include <sys/time.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string>
```

Include dependency graph for server.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct micliente
- class server

Index